q2_1

November 17, 2019

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import string
from sklearn.model_selection import KFold
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score, precision_score, recall_score, accuracy_score, c
from statistics import mean
from collections import Counter
from copy import deepcopy
import operator
from math import log2
from sklearn.model_selection import train_test_split, KFold

%matplotlib inline
```

In [2]:
```
df = pd.read_csv('ensemble_data.csv')
df.head()
```

Out[2]:

	type	cap_shape	cap_surface	cap_color	bruises	odor	gill_attachment	\
0	p	x	s	n	t	p	f	
1	e	x	s	y	t	a	f	
2	e	b	s	w	t	l	f	
3	p	x	y	w	t	p	f	
4	e	x	s	g	f	n	f	

	gill_spacing	gill_size	gill_color	...	stalk_surface_below_ring	\
0	c	n	k	...	s	
1	c	b	k	...	s	
2	c	b	n	...	s	
3	c	n	n	...	s	
4	w	b	k	...	s	

	stalk_color_above_ring	stalk_color_below_ring	veil_type	veil_color	\
0	w	w	p	w	
1	w	w	p	w	
2	w	w	p	w	

```
3                     w                       w         p          w
4                     w                       w         p          w

  ring_number ring_type spore_print_color population habitat
0           o         p                 k          s       u
1           o         p                 n          n       g
2           o         p                 n          n       m
3           o         p                 k          s       u
4           o         e                 n          a       g

[5 rows x 23 columns]
```

In [3]:
```
Y = df.iloc[:, 0].values
X = df
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=
```

In [4]:
```
header = ['Label', 'Text', 'Length', 'Unigram', 'Bigram', 'Trigram']
class Question:
    def __init__(self, column, value):
        self.column = column
        self.value = value

    def match(self, example):
        val = example[self.column]
        return self.value == val

    def __repr__(self):
        condition = "contains"
        return "Does %s %s %s?" % (
            "Col" + str(self.column), condition, str(self.value))
```

In [5]:
```
def class_counts(rows):
    counts = {}
    for row in rows:
        label = row[0]
        if label not in counts:
            counts[label] = 0
        counts[label] += 1
    return counts
```

In [6]:
```
def gini(rows):
    counts = class_counts(rows)
    impurity = 1
    for lbl in counts:
        prob_of_lbl = counts[lbl] / float(len(rows))
        impurity -= prob_of_lbl**2
    return impurity
```

```
In [7]: def info_gain(left, right, current_uncertainty, func=gini):
            p = float(len(left)) / (len(left) + len(right))
            return current_uncertainty - p * func(left) - (1 - p) * func(right)

In [8]: class Leaf:
            def __init__(self, rows):
                self.predictions = class_counts(rows)

In [9]: class Decision_Node:
            def __init__(self,
                         question,
                         true_branch,
                         false_branch):
                self.question = question
                self.true_branch = true_branch
                self.false_branch = false_branch

In [10]: def partition(rows, question):
             trueRows = []
             falseRows = []

             for r in rows:
                 if question.match(r):
                     trueRows.append(r)
                 else:
                     falseRows.append(r)

             return trueRows, falseRows

In [11]: def findBestSplit(rows, questions, func):
             best_gain = 0
             best_question = None
             current_uncertainty = func(rows)

             for q in questions:
                 trueRows, falseRows = partition(rows, q)
                 if len(trueRows) == 0 or len(falseRows) == 0:
                     continue

                 gain = info_gain(trueRows, falseRows, current_uncertainty, func)

                 if gain >= best_gain:
                     best_gain, best_question = gain, q

             return best_gain, best_question

In [12]: def formTree(rows, questions, func):
             gain, question = findBestSplit(rows, questions, func)
```

```
            if gain == 0:
                return Leaf(rows)

            trueRows, falseRows = partition(rows, question)
            questions.remove(question)

            trueBranch = formTree(trueRows, questions, func)
            falseBranch = formTree(falseRows, questions, func)

            return Decision_Node(question, trueBranch, falseBranch)
```

In [13]:
```
def classifyRow(node, row):
    if isinstance(node, Leaf):
        return node.predictions

    if node.question.match(row):
        return classifyRow(node.true_branch, row)
    else:
        return classifyRow(node.false_branch, row)
```

In [14]:
```
def train(data, questions, func):
    return formTree(data, deepcopy(questions), func)
```

In [15]:
```
def classify(root, rows):
    predictions = []
    for r in rows:
        predictions.append(max(classifyRow(root, r).items(), key=operator.itemgetter(
    return predictions
```

In [16]:
```
def getDataInIndex(data, index):
    l = []
    for i in range(len(data)):
        if i in index:
            l.append(data[i])
    return l
```

In [17]:
```
def getActualLabels(act_data):
    act_labels = []
    for d in act_data:
        act_labels.append(d[0])
    return act_labels
```

In [18]:
```
def get_unique_vals(X_train, index):
    ans = []
    for i, r in X_train.iterrows():
#         print(r)
        ans.append(r[index])

    return set(ans)
```

```
In [19]: questions = []

         for i in range(1, X_train.shape[1]):
             unique_vals = get_unique_vals(X_train,int(i))
             for val in unique_vals:
                 questions.append(Question(i,val))

In [20]: len(questions)

Out[20]: 117

In [21]: db_train = X_train.values.tolist()
         db_test = X_test.values.tolist()

In [22]: kfold = KFold(5, True, 1)
         precision = []
         recall = []
         f_score = []
         accuracy = []
         i = 0

         for trainInd,testInd in kfold.split(db_train):
             train_data = getDataInIndex(db_train, trainInd)
             test_data = getDataInIndex(db_train, testInd)

             root = train(train_data, questions, gini)

             prediction = classify(root, test_data)

             actual = getActualLabels(test_data)
             predicted = prediction

             precision.append(precision_score(actual, predicted, average='macro'))
             recall.append(recall_score(actual, predicted, average='macro'))
             f_score.append(f1_score(actual, predicted, average='macro'))
             accuracy.append(accuracy_score(actual, predicted))

             print("Training...")

         print("Precision Score = " + str(mean(precision)))
         print("Recall Score = " + str(mean(recall)))
         print("F Score = " + str(mean(f_score)))
         print("Accuracy = " + str(mean(accuracy)))

Training...
Training...
Training...
Training...
Training...
```

```
Precision Score = 0.9997093023255814
Recall Score = 0.9996742671009772
F Score = 0.9996913074553999
Accuracy = 0.9996923076923077
```