q1

November 17, 2019

In [219]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split, KFold
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.metrics import precision_score, accuracy_score, recall_score, f1_score
from statistics import mean

%matplotlib inline
```

In [204]:
```
df = pd.read_csv('ensemble_data.csv')
df.head()
```

Out[204]:

	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 [221]: len(df)
```

```
Out[221]: 8124
```

```
In [223]: labels = df.iloc[:, 0].values
```

```
In [224]: onehot_encoder = LabelEncoder()
          # labels = labels.reshape(len(labels), 1)
          labels = onehot_encoder.fit_transform(labels)
          print(labels)
```

```
[1 0 0 ... 0 1 0]
```

```
In [225]: df_2 = df.drop(columns=['type'])
          df_2 = pd.get_dummies(df_2,drop_first=False)
          df_2
```

```
Out[225]:       cap_shape_b  cap_shape_c  cap_shape_f  cap_shape_k  cap_shape_s  \
          0               0            0            0            0            0
          1               0            0            0            0            0
          2               1            0            0            0            0
          3               0            0            0            0            0
          4               0            0            0            0            0
          ...           ...          ...          ...          ...          ...
          8119            0            0            0            1            0
          8120            0            0            0            0            0
          8121            0            0            1            0            0
          8122            0            0            0            1            0
          8123            0            0            0            0            0

                cap_shape_x  cap_surface_f  cap_surface_g  cap_surface_s  cap_surface_y  \
          0               1              0              0              1              0
          1               1              0              0              1              0
          2               0              0              0              1              0
          3               1              0              0              0              1
          4               1              0              0              1              0
          ...           ...            ...            ...            ...            ...
```

```
8119           0              0              0              1              0
8120           1              0              0              1              0
8121           0              0              0              1              0
8122           0              0              0              0              1
8123           1              0              0              1              0

      ...  population_s  population_v  population_y  habitat_d  habitat_g  \
0     ...             1             0             0          0          0
1     ...             0             0             0          0          1
2     ...             0             0             0          0          0
3     ...             1             0             0          0          0
4     ...             0             0             0          0          1
...   ...           ...           ...           ...        ...        ...
8119  ...             0             0             0          0          0
8120  ...             0             1             0          0          0
8121  ...             0             0             0          0          0
8122  ...             0             1             0          0          0
8123  ...             0             0             0          0          0

      habitat_l  habitat_m  habitat_p  habitat_u  habitat_w
0             0          0          0          1          0
1             0          0          0          0          0
2             0          1          0          0          0
3             0          0          0          1          0
4             0          0          0          0          0
...         ...        ...        ...        ...        ...
8119          1          0          0          0          0
8120          1          0          0          0          0
8121          1          0          0          0          0
8122          1          0          0          0          0
8123          1          0          0          0          0

[8124 rows x 117 columns]
```

In [226]:
```
svm_mod = svm.SVC(gamma='scale', probability=True)
#mlp_mod = MLPClassifier(solver='adam', alpha=1e-5, hidden_layer_sizes=(5, 3, 2), ra
destr_mod = DecisionTreeClassifier(random_state=0)
rf_mod = RandomForestClassifier(n_estimators=10, max_depth=121, random_state=0)
lr_mod = LogisticRegression(random_state=0, solver='lbfgs', multi_class='auto')
```

In [227]:
```
df_2 = df_2.iloc[:,:].values
X_train, X_test, y_train, y_test = train_test_split(df_2, labels, test_size=0.2, ran
```

In [228]:
```
def getActualLabels(act_data):
    act_labels = []
    for d in act_data:
        act_labels.append(d[1])
    return act_labels
```

```
In [229]: def getDataInIndex(data, index):
              l = []
              for i in range(len(data)):
                  if i in index:
                      l.append(data[i])
              return l

In [230]: def appendProbs(cur, toAppend):
              ans = []
              for i in range(len(toAppend)):
                  c = toAppend[i]
                  row = c

                  if len(cur) > 0:
                      for a in cur[i]:
                          row.append(a)

                  ans.append(row)

              return ans

In [231]: def getActualLabels(data, ind):
              ans = []
              for i in range(len(data)):
                  if i in ind:
                      ans.append(data[i])

              return ans

In [232]: def forwardPassFirst(data, models):
              predDB = []
              ans = None
              for m in models:
                  y = m.predict_proba(data)

                  if ans is None:
                      ans = y
                  else:
                      ans = np.concatenate((ans, y), axis=1)

              return ans

In [233]: def mainMethod(base_mod_lis, base_mod_str, meta):

              fin_precision = {}
              fin_recall = {}
              fin_f_score = {}
              fin_acc = {}
              finalPredDB = None
```

```
finalLabelDB = None
kfold = KFold(5, True, 1)

for trainInd,testInd in kfold.split(X_train):

    train_data = getDataInIndex(X_train, trainInd)
    test_data = getDataInIndex(X_train, testInd)
    train_labels = getActualLabels(labels, trainInd)
    test_labels = getActualLabels(labels, testInd)
    predDB = None
    metaTestDB = None
    metaLabelDB = None

    for i in range(len(base_mod_lis)):
        mod = base_mod_lis[i]
        modStr = base_mod_str[i]

        mod.fit(train_data, train_labels)
        prob_classes_test = mod.predict_proba(test_data)

        if metaTestDB is not None:
            metaTestDB = np.concatenate((metaTestDB, prob_classes_test), axis=1)
        else:
            metaTestDB = prob_classes_test

        predicted = mod.predict(test_data)

        if modStr not in fin_precision:
            fin_precision[modStr] = []
            fin_recall[modStr] = []
            fin_f_score[modStr] = []
            fin_acc[modStr] = []

        fin_precision[modStr].append(precision_score(test_labels, predicted, labe
        fin_recall[modStr].append(recall_score(test_labels, predicted, labels=np
        fin_f_score[modStr].append(f1_score(test_labels, predicted, labels=np.uni
        fin_acc[modStr].append(accuracy_score(test_labels, predicted))


    if finalPredDB is None:
        finalPredDB = metaTestDB
        finalLabelDB = test_labels
    else:
        finalPredDB = np.concatenate((finalPredDB, metaTestDB))
        finalLabelDB.extend(test_labels)

    print("End of fold...")
```

```
                    print("\nAccuracy Scores of Base Layer Models")

                    for i in range(len(base_mod_lis)):
                        print(base_mod_str[i] + ":" + str(mean(fin_acc[base_mod_str[i]])))

                    print("\nPrecision Scores of Base Layer Models")

                    for i in range(len(base_mod_lis)):
                        print(base_mod_str[i] + ":" + str(mean(fin_precision[base_mod_str[i]])))

                    print("\nRecall Scores of Base Layer Models")

                    for i in range(len(base_mod_lis)):
                        print(base_mod_str[i] + ":" + str(mean(fin_recall[base_mod_str[i]])))

                    print("\nF Scores of Base Layer Models")

                    for i in range(len(base_mod_lis)):
                        print(base_mod_str[i] + ":" + str(mean(fin_f_score[base_mod_str[i]])))

                    meta.fit(finalPredDB, finalLabelDB)
                    firstLayerOutput = forwardPassFirst(X_test, base_mod_lis)
                    y_pred = meta.predict(firstLayerOutput)

                    return y_pred
```

```
In [234]: lis = [svm_mod, destr_mod, rf_mod, lr_mod]
          mod_str = ["SVM", "DESTR", "RF_MOD", "LR_MOD"]
          y_pred = []
          for model in range(len(lis)):
              y_pred.append(mainMethod(lis[:model]+lis[model+1:], mod_str[:model]+mod_str[model
```

```
End of fold...

Accuracy Scores of Base Layer Models
DESTR:0.5
RF_MOD:0.5169230769230769
LR_MOD:0.5507692307692308

Precision Scores of Base Layer Models
DESTR:0.4175627240143369
RF_MOD:0.4107142857142857
LR_MOD:0.40441176470588236

Recall Scores of Base Layer Models
DESTR:0.4175627240143369
RF_MOD:0.2885304659498208
LR_MOD:0.0985663082437276
```

```
F Scores of Base Layer Models
DESTR:0.41756272401433686
RF_MOD:0.33894736842105266
LR_MOD:0.1585014409221902
End of fold...

Accuracy Scores of Base Layer Models
SVM:0.5607692307692308
RF_MOD:0.5169230769230769
LR_MOD:0.5507692307692308

Precision Scores of Base Layer Models
SVM:0.3673469387755102
RF_MOD:0.4107142857142857
LR_MOD:0.40441176470588236

Recall Scores of Base Layer Models
SVM:0.03225806451612903
RF_MOD:0.2885304659498208
LR_MOD:0.0985663082437276

F Scores of Base Layer Models
SVM:0.05930807248764415
RF_MOD:0.33894736842105266
LR_MOD:0.1585014409221902
End of fold...

Accuracy Scores of Base Layer Models
SVM:0.5607692307692308
DESTR:0.5
LR_MOD:0.5507692307692308

Precision Scores of Base Layer Models
SVM:0.3673469387755102
DESTR:0.4175627240143369
LR_MOD:0.40441176470588236

Recall Scores of Base Layer Models
SVM:0.03225806451612903
DESTR:0.4175627240143369
LR_MOD:0.0985663082437276

F Scores of Base Layer Models
SVM:0.05930807248764415
DESTR:0.41756272401433686
LR_MOD:0.1585014409221902
End of fold...
```

```
Accuracy Scores of Base Layer Models
SVM:0.5607692307692308
DESTR:0.5
RF_MOD:0.5169230769230769

Precision Scores of Base Layer Models
SVM:0.3673469387755102
DESTR:0.4175627240143369
RF_MOD:0.4107142857142857

Recall Scores of Base Layer Models
SVM:0.03225806451612903
DESTR:0.4175627240143369
RF_MOD:0.2885304659498208

F Scores of Base Layer Models
SVM:0.05930807248764415
DESTR:0.41756272401433686
RF_MOD:0.33894736842105266
```

```
In [241]: for predic in range(len(y_pred)):
              a = accuracy_score(y_test, y_pred[predic])
              print('Accuracy for '+mod_str[predic]+' as meta = '+str(a))
```

```
Accuracy for SVM as meta = 0.5187692307692308
Accuracy for DESTR as meta = 0.47507692307692306
Accuracy for RF_MOD as meta = 0.4763076923076923
Accuracy for LR_MOD as meta = 0.5187692307692308
```