Metabolism_final-Copy1

February 1, 2018

0.0.1 Load the metabolisam classification

Data file includes the microbial nomenclature. Load the names and metabolic classification into a python dictionary data structure so that the name can be used to look up the metabolic classification File is tab delimitated.

Data file example:

```
In [1]: from sklearn.tree import DecisionTreeClassifier
        from sklearn.model_selection import train_test_split
        from sklearn.feature_selection import SelectKBest
        from sklearn.cross_validation import train_test_split
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import classification_report
        from sklearn import feature_selection
        from sklearn.metrics import recall_score
        from sklearn.model_selection import StratifiedKFold
        from sklearn.linear_model import LogisticRegression
        import seaborn; seaborn.set()
        import matplotlib.pyplot as plt
        %matplotlib inline

        organisams={}
        organism_id=[]
        attributes_full={}
        atributes_by_class={}
        class_set=set()
        class_counter = {'anaerobic':0, 'aerobic':0, 'facultative':0}
        with open("aviFolder/cleanData.txt") as infile:
            for line in infile:
                # data is tab delimited
                data = line.rstrip().split("\t")
                # first column contains organism name
                organism_id.append(data[0])
                # second column contains metabolic class
                organisams[data[0]]={"class":data[1],"roles":[]}
                class_set.add(data[1].replace("'", ""))
                classtype=data[1].replace("'", "")
                class_counter[classtype]+=1
```

```
                for i in range(2,len(data)):
                    attribute = data[i].replace("'", "")
                    if(attribute not in organisams[data[0].replace("'", "")]["roles"]):
                        organisams[data[0]]["roles"].append(attribute)
                        if not attribute in attributes_full:
                            attributes_full[attribute]=0
                            atributes_by_class[attribute]={'anaerobic':0, 'aerobic':0, 'facultat
                        attributes_full[attribute]+=1
                        atributes_by_class[attribute][classtype]+=1
```

```
/home/ric/Environments/notebook/lib/python3.6/site-packages/sklearn/cross_validation.py:41: Depr
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
In [2]: print(len(organisams))
        print(len(attributes_full))
        print(class_set)
        print(class_counter)
```

```
290
127711
{'anaerobic', 'aerobic', 'facultative'}
{'anaerobic': 45, 'aerobic': 214, 'facultative': 31}
```

A one dimensional array is constructed for the organisms and all of the functional roles. This is used to create the functional role data matrix; full_attribute_array

```
In [3]: import numpy
        attribute_list = list(attributes_full.keys())
        organisam_list = list(organisams.keys())
        class_list = list(class_set)
        full_classification_array = numpy.array(["" for x in range(len(organisam_list))])

        print(class_list)
        print(len(attributes_full))
        print(len(organisam_list))
        print(len(full_classification_array))

        full_attribute_array = numpy.zeros(shape=(len(organisam_list),len(attribute_list)))
```

```
['anaerobic', 'aerobic', 'facultative']
127711
290
290
```

```
In [4]: for y_indx in range(len(organisam_list)):
            organisam = organisam_list[y_indx]
            if organisam in organisams:
                full_classification_array[y_indx] = class_list.index(organisams[organisam]["clas
                for attribute in organisams[organisam]["roles"]:
                    if attribute in attributes_full:
                        x_indx  = attribute_list.index(attribute)
                        full_attribute_array[y_indx,x_indx]=1
```

0.0.2 Learning curve

```
In [5]: train_index=[]
        test_index=[]
        splits =10
        skf = StratifiedKFold(n_splits=splits,random_state=0)
        for train_idx, test_idx in skf.split(full_attribute_array,full_classification_array):
            train_index.append(train_idx)
            test_index.append(test_idx)
```

```
In [6]: import seaborn as sns
        def plot_confusion_matrix(cm, classes, title,classifier_name):
            plt.rcParams.update({'font.size': 22})
            fig,ax= plt.subplots(figsize=(6,5))
            sns.set(font_scale=1.5)
            sns_plot = sns.heatmap(cm, annot=True, ax = ax, cmap="Blues"); #annot=True to annota
            # labels, title and ticks
            ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
            ax.set_title(title);
            ax.xaxis.set_ticklabels(classes); ax.yaxis.set_ticklabels(classes);
            fig.savefig(classifier_name+".png")
```

```
In [7]: def cf_stats(TN,TP,FP,FN):
            AN = TN+FP
            AP = TN+FN
            PN = TN+FN
            PP = TP+FP
            Total = TN+TP+FP+FN
            Recall = (TP/(TP+FN))
            Precision = (TP/(TP+FP))
            #print("TN: %6.3f\t TP: %6.3f\t FP: %6.3f\t FN: %6.3f\t "%(TN,TP,FP,FN))
            #print("Total: %6.3f\t"%(Total))
            print("Accuracy:\t\t%6.3f"%((TP+TN)/Total))
            #print("Misclassification Rate:\t%6.3f"%((FN+FP)/Total))
            #print("True Positive Rate:\t%6.3f"%(TP/Total))
            #print("True Negative Rate:\t%6.3f"%(TN/Total))
            #print("False Positive Rate:\t%6.3f"%(FP/Total))
```

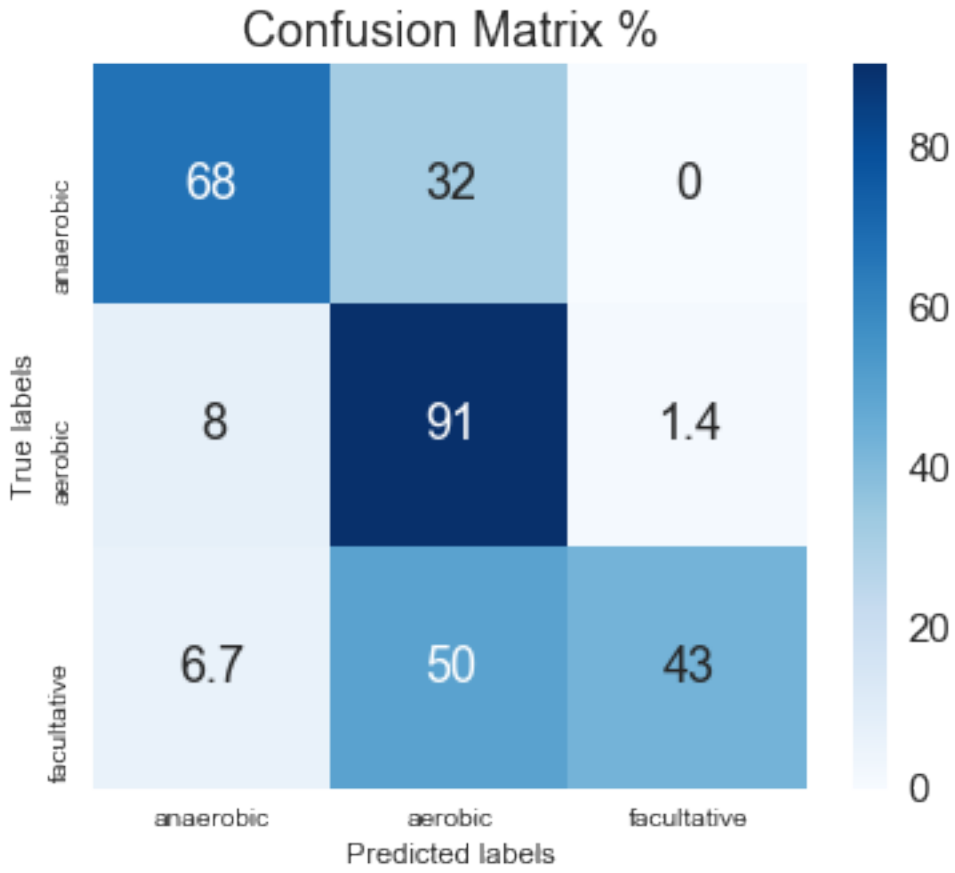
```
            #print("False Negative Rate:\t%6.3f"%(FN/Total))
            #print("Specificity:\t\t%6.3f"%(TN/AN))
            print("Precision:\t\t%6.3f"%(Precision))
            #print("Prevalence:\t\t%6.3f"%(AP/Total))
            print("Recall:\t\t%6.3f"%(Recall))
            print("F1 score::\t\t%6.3f"%(2*((Precision*Recall)/(Precision+Recall))))
            print()

In [8]: def classiferTest(classifier,classifier_name,print_cfm):
            if print_cfm:
                print(classifier_name)
            train_score = numpy.zeros(splits)
            validate_score = numpy.zeros(splits)
            cnf_matrix = numpy.zeros(shape=(3,3))
            cnf_matrix_f = numpy.zeros(shape=(3,3))
            for c in range(splits):
                X_train = full_attribute_array[train_index[c]]
                y_train = full_classification_array[train_index[c]]
                X_test = full_attribute_array[test_index[c]]
                y_test = full_classification_array[test_index[c]]
                classifier.fit(X_train,y_train)
                train_score[c] = classifier.score(X_train,y_train)
                validate_score[c] = classifier.score(X_test,y_test)
                y_pred = classifier.predict(X_test)
                cnf = confusion_matrix(y_test, y_pred)
                cnf_f = cnf.astype('float') / cnf.sum(axis=1)[:, numpy.newaxis]
                for i in range(len(cnf)):
                    for j in range(len(cnf)):
                        cnf_matrix[i][j] += cnf[i][j]
                        cnf_matrix_f[i][j] += cnf_f[i][j]

            print("%6.3f\t%6.3f\t%6.3f\t%6.3f" % (numpy.average(train_score),numpy.std(train_sco


            if print_cfm:
                cnf_av = cnf_matrix/splits
                print()
                print(cnf_av[0][0],cnf_av[0][1],cnf_av[0][2],)
                print(cnf_av[1][0],cnf_av[1][1],cnf_av[1][2],)
                print(cnf_av[2][0],cnf_av[2][1],cnf_av[2][2],)
                print()
                print(class_list[0])
                TP = cnf_av[0][0]
                TN = cnf_av[1][2]+cnf_av[1][2]+cnf_av[2][1]+cnf_av[2][2]
                FP = cnf_av[0][1] + cnf_av[0][2]
                FN = cnf_av[1][0] + cnf_av[2][0]
                cf_stats(TN,TP,FP,FN)
```

```
            print(class_list[1])
            TP = cnf_av[1][1]
            TN = cnf_av[0][0]+cnf_av[0][2]+cnf_av[2][0]+cnf_av[2][2]
            FP = cnf_av[1][0] + cnf_av[1][2]
            FN = cnf_av[0][1] + cnf_av[2][1]
            cf_stats(TN,TP,FP,FN)

            print(class_list[2])
            TP = cnf_av[2][2]
            TN = cnf_av[0][0]+cnf_av[0][1]+cnf_av[1][0]+cnf_av[1][1]
            FP = cnf_av[2][0] + cnf_av[2][1]
            FN = cnf_av[0][1] + cnf_av[0][2]
            cf_stats(TN,TP,FP,FN)

            print(classifier)
            print()
            print("Confusion matrix")
            for i in range(len(cnf_matrix)):
                print(class_list[i],end="  \t")
                for j in range(len(cnf_matrix[i])):
                    print(cnf_matrix[i][j]/splits,end="\t")
                print()
            print()
            for i in range(len(cnf_matrix_f)):
                print(class_list[i],end="  \t")
                for j in range(len(cnf_matrix_f[i])):
                    print("%6.1f" %((cnf_matrix_f[i][j]/splits)*100.0),end="\t")
                print()
            print()
            print("01",cnf_matrix[0][1])

            ##plot_confusion_matrix(cnf_matrix/10,class_list,'Confusion Matrix')
            plot_confusion_matrix(cnf_matrix_f/10*100.0,class_list,'Confusion Matrix %',clas
        return (numpy.average(train_score),numpy.std(train_score),numpy.average(validate_sco
```

In [9]:
```
from sklearn.neighbors import KNeighborsClassifier
classiferTest(KNeighborsClassifier(),"Metabolism-KNeighborsClassifier",True)
```

```
Metabolism-KNeighborsClassifier
 0.919         0.009         0.820         0.125

3.0 1.5 0.0
1.7 19.4 0.3
0.2 1.6 1.3

anaerobic
Accuracy:                 0.657
Precision:                 0.667
```

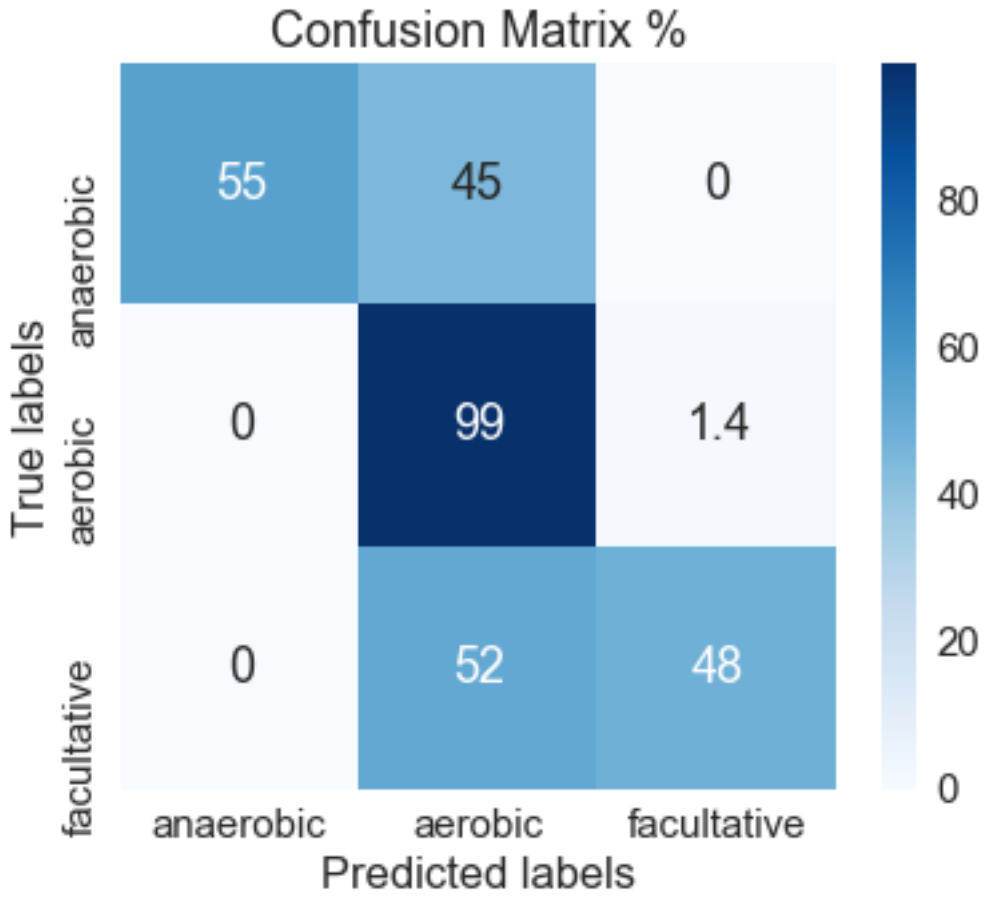
```
Recall:                 0.612
F1 score::                 0.638

aerobic
Accuracy:                0.824
Precision:                 0.907
Recall:                 0.862
F1 score::                 0.884

facultative
Accuracy:                0.891
Precision:                 0.419
Recall:                 0.464
F1 score::                 0.441

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')

Confusion matrix
anaerobic          3.0        1.5        0.0
aerobic          1.7        19.4        0.3
facultative          0.2        1.6        1.3

anaerobic            67.5          32.5          0.0
aerobic             8.0          90.6          1.4
facultative             6.7          50.0          43.3

01 15.0


Out[9]: (0.91915862663296211,
         0.0094682083187915475,
         0.81981911118173634,
         0.12502641484085469)
```




```
In [10]: from sklearn.naive_bayes import GaussianNB
         classiferTest(GaussianNB(),"Metabolism-GaussianNB",True)

Metabolism-GaussianNB
 1.000          0.000          0.867          0.071

2.5 2.0 0.0
0.0 21.1 0.3
0.0 1.6 1.5

anaerobic
Accuracy:                  0.756
Precision:                  0.556
Recall:                   1.000
F1 score::                  0.714

aerobic
Accuracy:                  0.866
Precision:                  0.986
```

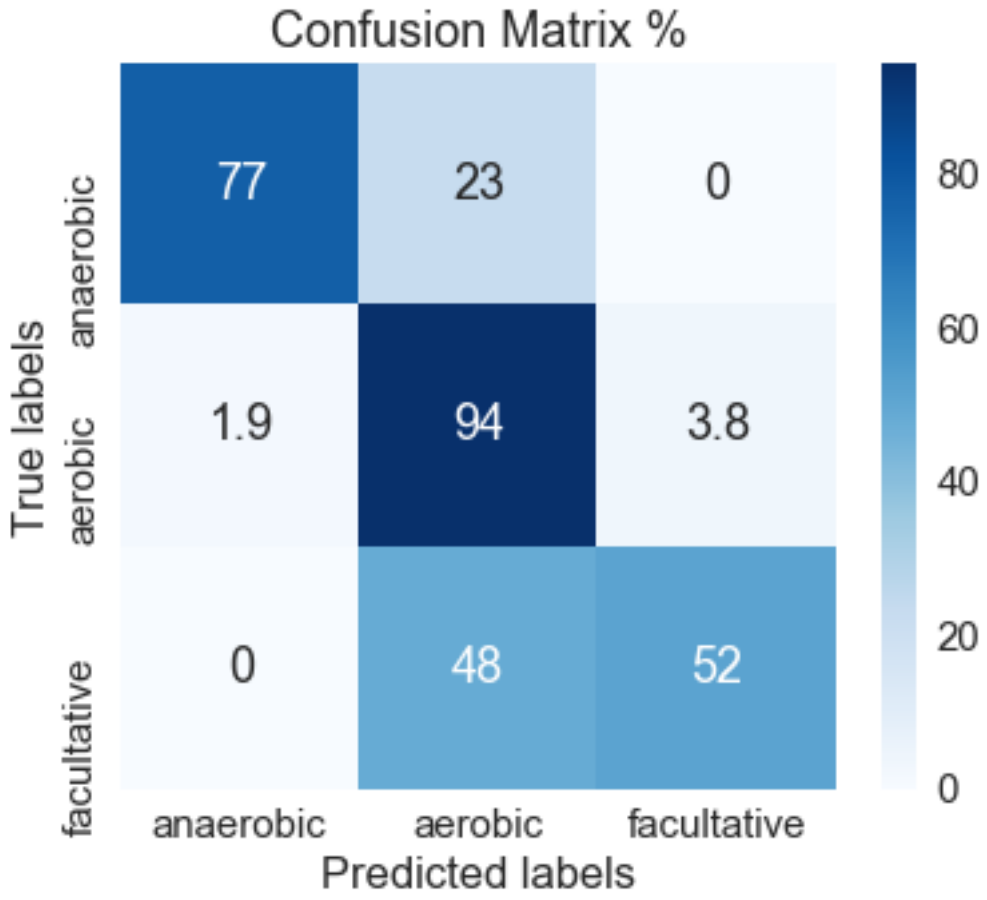
```
Recall:                 0.854
F1 score::                 0.915

facultative
Accuracy:                0.883
Precision:                 0.484
Recall:                 0.429
F1 score::                 0.455

GaussianNB(priors=None)

Confusion matrix
anaerobic          2.5        2.0        0.0
aerobic          0.0        21.1        0.3
facultative          0.0        1.6        1.5

anaerobic            55.0          45.0          0.0
aerobic             0.0          98.6          1.4
facultative             0.0          51.7          48.3

01 20.0
```

Out[10]: (1.0, 0.0, 0.86659833677631237, 0.071412628824689239)




```
In [11]: classiferTest(LogisticRegression(random_state=0),"Metabolism-LogisticRegression",True)

Metabolism-LogisticRegression
 1.000          0.000          0.871          0.103

3.4 1.1 0.0
0.4 20.2 0.8
0.0 1.5 1.6

anaerobic
Accuracy:                  0.844
Precision:                  0.756
Recall:                 0.895
F1 score::                  0.819

aerobic
Accuracy:                  0.869
Precision:                  0.944
Recall:                 0.886
```

```
F1 score::                 0.914

facultative
Accuracy:                 0.911
Precision:                 0.516
Recall:                0.593
F1 score::                 0.552

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=0, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)

Confusion matrix
anaerobic          3.4        1.1        0.0
aerobic          0.4        20.2        0.8
facultative          0.0        1.5        1.6

anaerobic            77.0          23.0          0.0
aerobic             1.9          94.3          3.8
facultative             0.0          48.3          51.7

01 11.0


Out[11]: (1.0, 0.0, 0.87080936490280203, 0.1025534374480245)
```

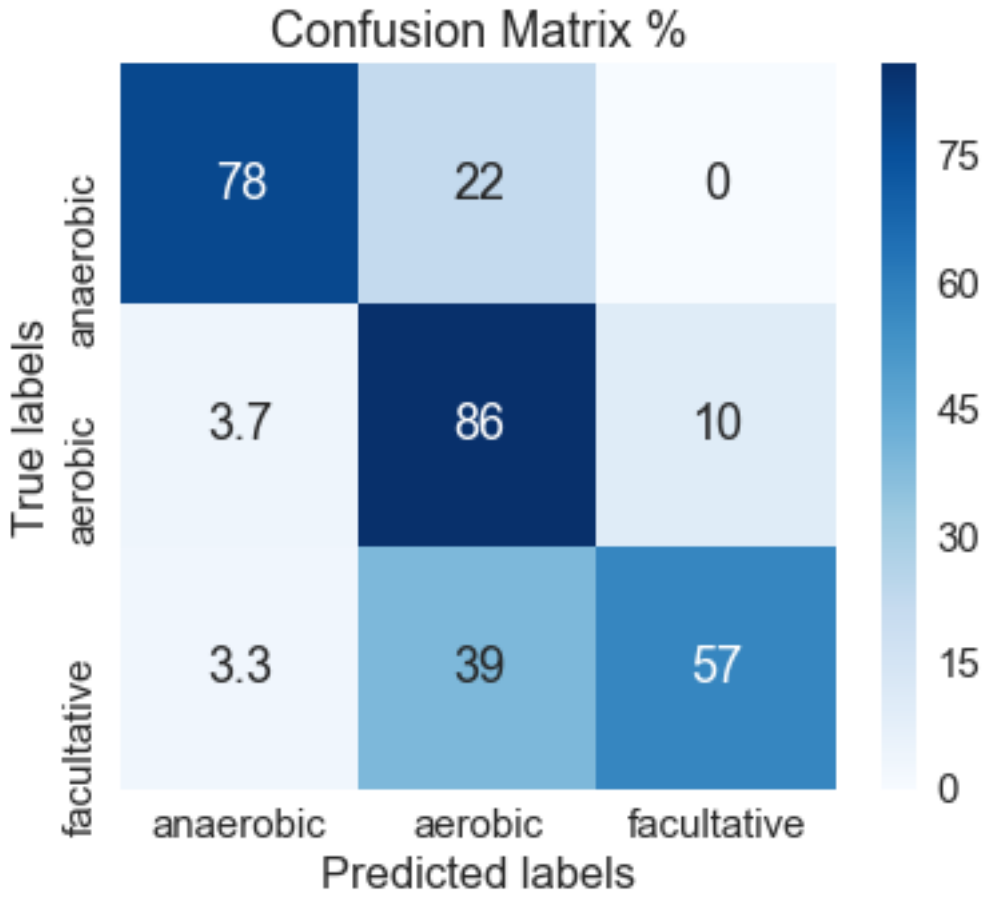



```
In [12]: classiferTest(DecisionTreeClassifier(random_state=0),"Metabolism-DecisionTreeClassifier
```

```
Metabolism-DecisionTreeClassifier
 1.000          0.000          0.818          0.066

3.5 1.0 0.0
0.8 18.4 2.2
0.1 1.2 1.8

anaerobic
Accuracy:                  0.852
Precision:                  0.778
Recall:                 0.795
F1 score::                  0.787

aerobic
Accuracy:                  0.821
Precision:                  0.860
Recall:                 0.893
```

```
F1 score::                 0.876

facultative
Accuracy:                 0.917
Precision:                 0.581
Recall:                0.643
F1 score::                 0.610

DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')

Confusion matrix
anaerobic          3.5        1.0        0.0
aerobic          0.8        18.4        2.2
facultative          0.1        1.2        1.8

anaerobic            78.0          22.0           0.0
aerobic             3.7          86.0          10.3
facultative             3.3          39.2          57.5

01 10.0
```

Out[12]: (1.0, 0.0, 0.81812781397319778, 0.066419387835499943)

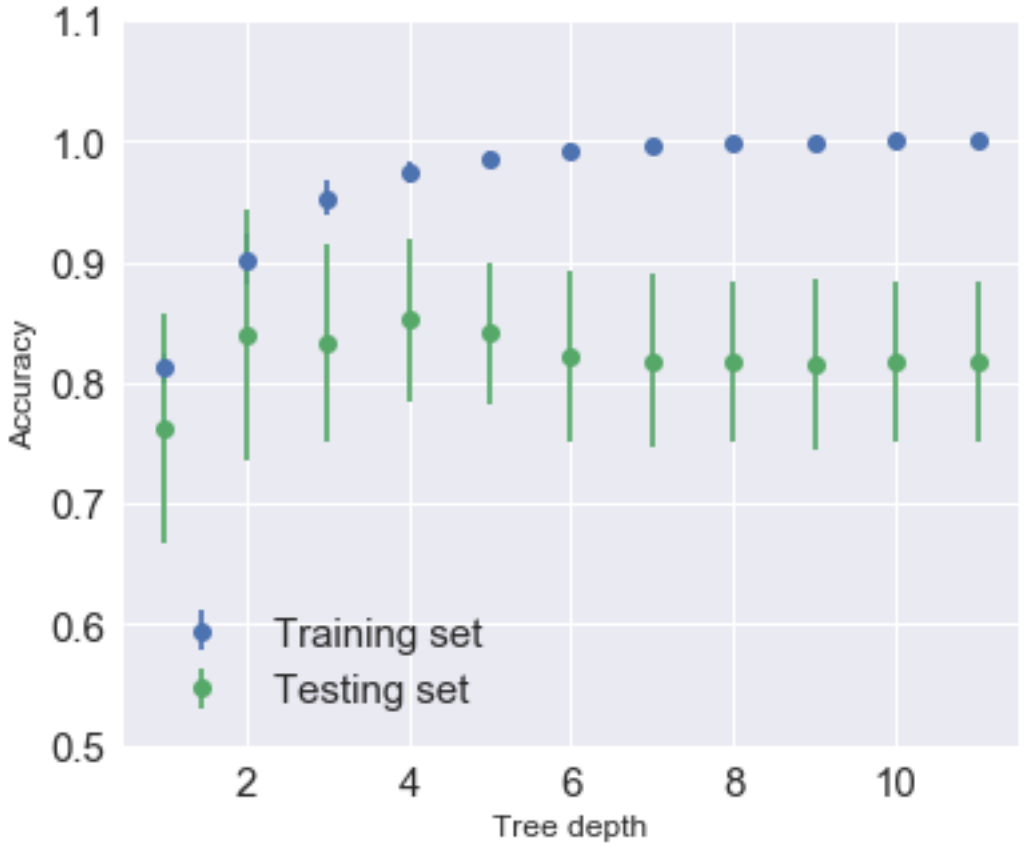



```
In [13]: val = numpy.zeros(12)
         test_av = numpy.zeros(12)
         test_std = numpy.zeros(12)
         val_av = numpy.zeros(12)
         val_std = numpy.zeros(12)
         for d in range(1,12):
             val[d]=d
             (test_av[d],test_std[d],val_av[d],val_std[d]) = classiferTest(DecisionTreeClassifie
```

```
0.812          0.012          0.762          0.095
0.902          0.021          0.840          0.104
0.953          0.014          0.832          0.081
0.974          0.010          0.852          0.068
0.985          0.007          0.842          0.059
0.991          0.006          0.822          0.070
0.996          0.005          0.818          0.071
0.998          0.003          0.818          0.066
0.999          0.002          0.815          0.071
```

```
 1.000          0.000          0.818          0.066
 1.000          0.000          0.818          0.066
```

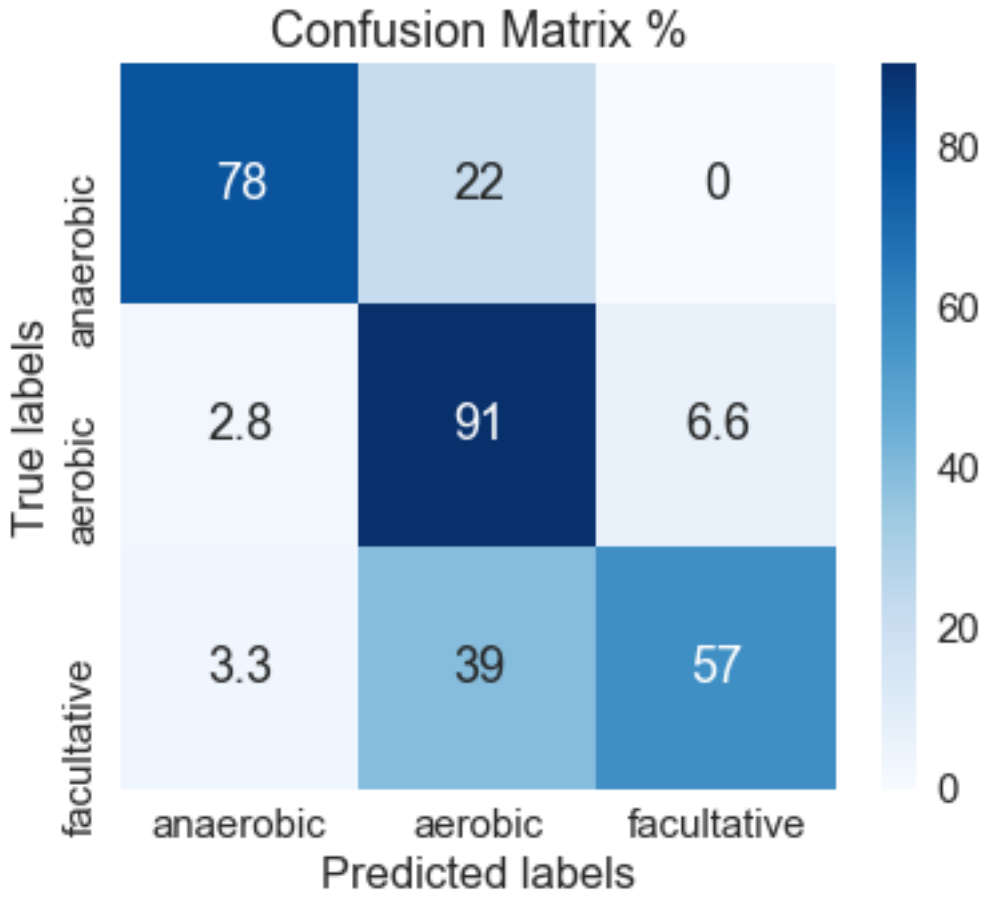
In [14]:
```
fig,ax = plt.subplots(figsize=(6,5))
plt.errorbar(val[1:],test_av[1:],yerr=test_std[1:],fmt='o',  label='Training set')
plt.errorbar(val[1:],val_av[1:],yerr=val_std[1:],fmt='o',label='Testing set')
ax.set_ylim(ymin=0.5,ymax=1.1)
plt.xlabel('Tree depth', fontsize=12)
plt.ylabel('Accuracy', fontsize=12)
plt.legend(loc='lower left')
plt.savefig("gini_depth-met.png")
```



In [ ]:
```
classiferTest(DecisionTreeClassifier(random_state=0,max_depth=4),"Metabolism-DecisionTr
```

```
Metabolism-DecisionTreeClassifier-gini
 0.974          0.010          0.852          0.068

3.5 1.0 0.0
0.6 19.4 1.4
```

```
0.1 1.2 1.8

anaerobic
Accuracy:                 0.845
Precision:                 0.778
Recall:                0.833
F1 score::                 0.805

aerobic
Accuracy:                 0.855
Precision:                 0.907
Recall:                0.898
F1 score::                 0.902

facultative
Accuracy:                 0.920
Precision:                 0.581
Recall:                0.643
F1 score::                 0.610

DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=4,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')

Confusion matrix
anaerobic          3.5        1.0        0.0
aerobic          0.6        19.4        1.4
facultative          0.1        1.2        1.8

anaerobic            78.0          22.0           0.0
aerobic             2.8          90.6           6.6
facultative             3.3          39.2          57.5

01 10.0


Out[ ]: (0.97397973574730601,
         0.0095820634648935595,
         0.85174532549393511,
         0.067604516943544704)
```




```
In [ ]: val = numpy.zeros(12)
        test_av = numpy.zeros(12)
        test_std = numpy.zeros(12)
        val_av = numpy.zeros(12)
        val_std = numpy.zeros(12)
        for d in range(1,12):
            val[d]=d
            (test_av[d],test_std[d],val_av[d],val_std[d])= classiferTest(DecisionTreeClassifier(
```

```
 0.812          0.010          0.786          0.106
 0.905          0.017          0.847          0.106
```

```
In [ ]: fig,ax = plt.subplots(figsize=(6,5))
        plt.errorbar(val[1:],test_av[1:],yerr=test_std[1:],fmt='o',  label='Training set')
        plt.errorbar(val[1:],val_av[1:],yerr=val_std[1:],fmt='o',label='Testing set')
        ax.set_ylim(ymin=0.5,ymax=1.1)
        plt.xlabel('Tree depth', fontsize=12)
        plt.ylabel('Accuracy', fontsize=12)
```

```
        plt.legend(loc='lower left')
        plt.savefig("entropy_depth-met.png")
```

```
In [ ]: classiferTest(DecisionTreeClassifier(random_state=0,max_depth=5, criterion='entropy'),"M
```

```
In [ ]: def get_code(tree, feature_names, target_names,
                       spacer_base="    "):
            """Produce psuedo-code for decision tree.
            Notes
            -----
            based on http://stackoverflow.com/a/30104792.
            """
            left      = tree.tree_.children_left
            right     = tree.tree_.children_right
            threshold = tree.tree_.threshold
            features  = [feature_names[i] for i in tree.tree_.feature]
            value = tree.tree_.value

            def recurse(left, right, threshold, features, node, depth):
                spacer = spacer_base * depth
                if (threshold[node] != -2):
                    print(spacer + "if ( " + features[node] + " <= " + \
                          str(threshold[node]) + " ) {")
                    if left[node] != -1:
                            recurse(left, right, threshold, features,
                                    left[node], depth+1)
                    print(spacer + "}\n" + spacer +"else {")
                    if right[node] != -1:
                            recurse(left, right, threshold, features,
                                    right[node], depth+1)
                    print(spacer + "}")
                else:
                    target = value[node]
                    for i, v in zip(numpy.nonzero(target)[1],
                                    target[numpy.nonzero(target)]):
                        target_name = target_names[i]
                        target_count = int(v)
                        print(spacer + "return " + str(target_name) + \
                              " ( " + str(target_count) + " examples )")

            recurse(left, right, threshold, features, 0, 0)
```

```
In [ ]: tree = DecisionTreeClassifier(random_state=0,max_depth=5, criterion='entropy')
        tree.fit(full_attribute_array,full_classification_array)
```

```
In [ ]: get_code(tree, attribute_list, class_list)
```

```
In [ ]: from sklearn.tree import export_graphviz
        import graphviz
```

```
        import os
        def printTree(tree,name):
            export_graphviz(tree, out_file="mytree.dot",feature_names=attribute_list,
                                class_names=class_list  )
            with open("mytree.dot") as f:
                dot_graph = f.read()
            os.system('dot -Tpng mytree.dot >  '+name+'.png ')
```

```
In [ ]: printTree(tree,"FullTreeClasifier")
```

```
In [ ]: train_score = tree.score(full_attribute_array,full_classification_array)
        y_pred = tree.predict(full_attribute_array)
        cnf = confusion_matrix(y_pred, full_classification_array)
```

```
In [ ]: for i in range(len(organism_id)):
            if y_pred[i]!=full_classification_array[i]:
                print(organism_id[i])
```