University of Regina

ENSE 865: Applied Machine Learning

PROGRAMMING ASSIGNMENT 4

SUBMITTED BY: Hitarthi Gandhi (200428713)

Question1. Predicting Sales using regression trees. You can use Scikit Learn for implementing trees.

Here, we are using caseates dataset to predicting sales and built regression trees. We used Scikit learn for implementing trees. First, we read csv file using pandas library. Then we assigned integer value to labels because in given dataset there is column name with 'Yes' and 'No'. Initially we importing all necessary python libraries.

Below code gives integer values to the labels.

```
Dataset.Sales = Dataset.Sales.map(lambda x: 0 if x<=8 else 1)

Dataset.ShelveLoc = pd.factorize(Dataset.ShelveLoc)[0]

Dataset.Urban = Dataset.Urban.map({'No':0, 'Yes':1})

Dataset.US = Dataset.US.map({'No':0, 'Yes':1})
```

(a) Split the data set into a training set and a test set.

Below code Spliting the dataset into 80%training data and 20% testing data.

```
X = Dataset.iloc[:,1:11]

y = Dataset['Sales']

X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, y, train_size=0.2,
random_state=18)
```

(b) Fit a regression tree to the training set. Plot the tree and interpret the results. What to test MSE do you obtain?

To build decision tree we are using scikit learn decisiontreeregressor. We fit a regression tree over the training dataset. Here we take decision tree max depth 4. Below code is fit a regression tree and plot a decision regression tree.

```
Regression_tree = DecisionTreeRegressor(max_depth=4)

Regression_tree.fit(X_Train, Y_Train)

Fig=plt.figure(figsize=(24,12))

tree.plot_tree(Regression_tree.fit(X_Train,
Y_Train),feature_names=X.columns,filled=True,rounded=True,fontsize=16);

plt.title('Decision Regression Tree');
```

Output:

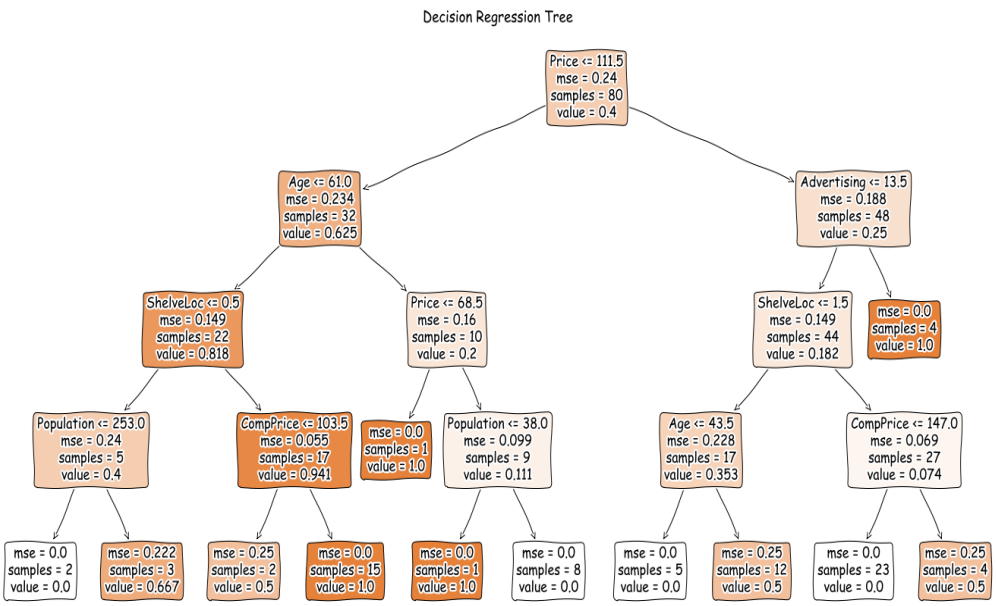

Fig1. Decision Regression Tree

Below code is perform prediction on test data and Calculate the mean squared error.

Prediction_R = Regression_tree.predict(X_Test)

plt.xkcd()

plt.figure(figsize=(24, 12))

plt.scatter(Prediction_R, Y_Test, label = 'medv', color='g')

plt.plot([0, 1], [0, 1], 'g', transform = plt.gca().transAxes)

plt.xlabel('Prediction', color='b', fontsize=22)

plt.ylabel('Test-Data', color='b', fontsize=22)

plt.title('Predicted Values Mean Squared Error', fontsize=25, color='b')

print("Mean Squared Error of regression tree: ", mean_squared_error(Y_Test, Prediction_R))

Output:

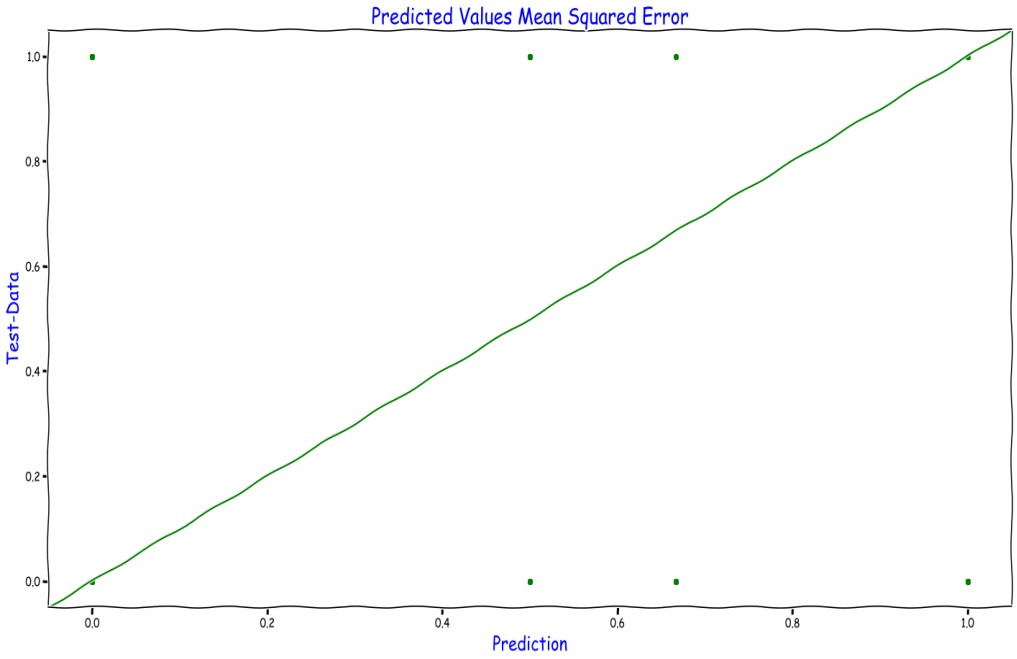

Fig2. Mean Squared Error Graph of Regressor Tree

(c) Use cross-validation in order to determine the optimal level of tree complexity. Does pruning the tree improve the test MSE?

Below code is perform cross validation to find the best leaf.

```
SCORES = []

Max_Leaf_Array = range(2, 50)

for Max_Leafs in Max_Leaf_Array:

    regressiontree = DecisionTreeRegressor(max_leaf_nodes=Max_Leafs)

    sc = cross_val_score(regressiontree, X, y, cv=6,
scoring="neg_mean_squared_error")

    SCORES.append((-sc.mean(), sc.std()))

SCORES = np.array(SCORES)
```

Output:

```
Number of best leafs are 11.
```

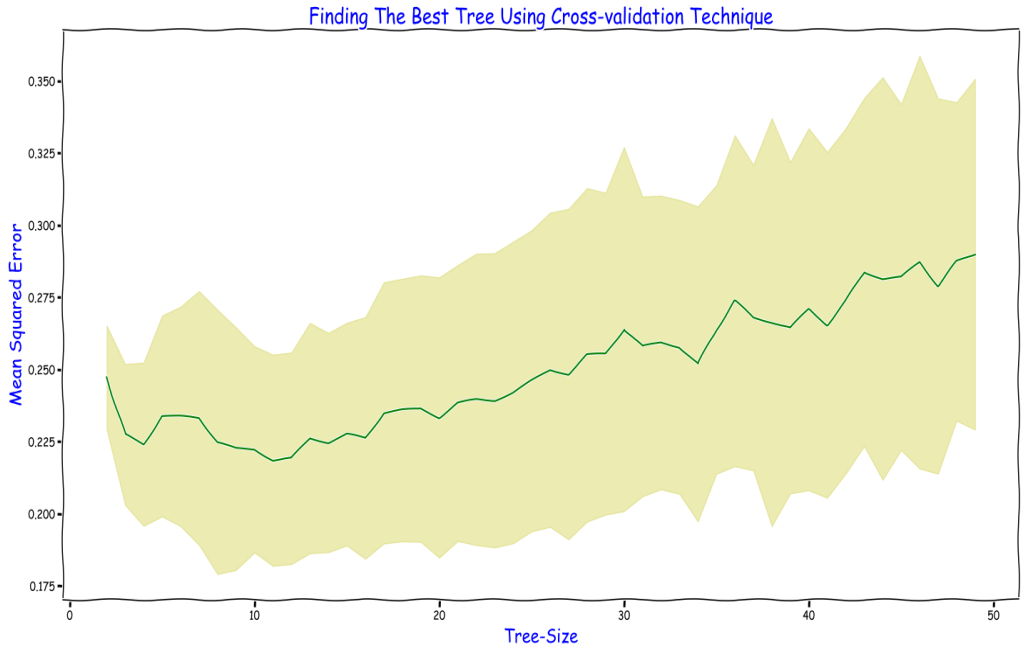

Fig3. Cross-Validation Technique for Finding the Best Tree

Pruning is a data compression technique which reduces the size of decision trees by removing parts of the tree that are non-critical and redundant to classify instances. Pruning reduce the complexity of model and improves the accuracy and decreases the overfitting. Overall, Pruning increases the classification error but it decreases the Mean squared error because it reduces the overfitting.

(d) Use the bagging approach in order to analyse this data. What test MSE do you obtain?

Below code is performing bagging approach and calculating test MSE.

Bagging= RandomForestRegressor(max_features=6).fit(X_Train, Y_Train)

Bagging_Prediction = Bagging.predict(X_Test)

plt.xkcd()

plt.figure(figsize=(24, 12))

plt.scatter(Bagging_Prediction, Y_Test, label = 'medv', color='r')

plt.plot([0, 1], [0, 1], 'g', transform = plt.gca().transAxes)

plt.xlabel('Predictions',fontsize=22, color='b')

plt.ylabel('Test-Data',fontsize=22, color='b')

print("Mean Squared Error USing Bagging Approach: ", mean_squared_error(Y_Test, Bagging_Prediction))

Output:

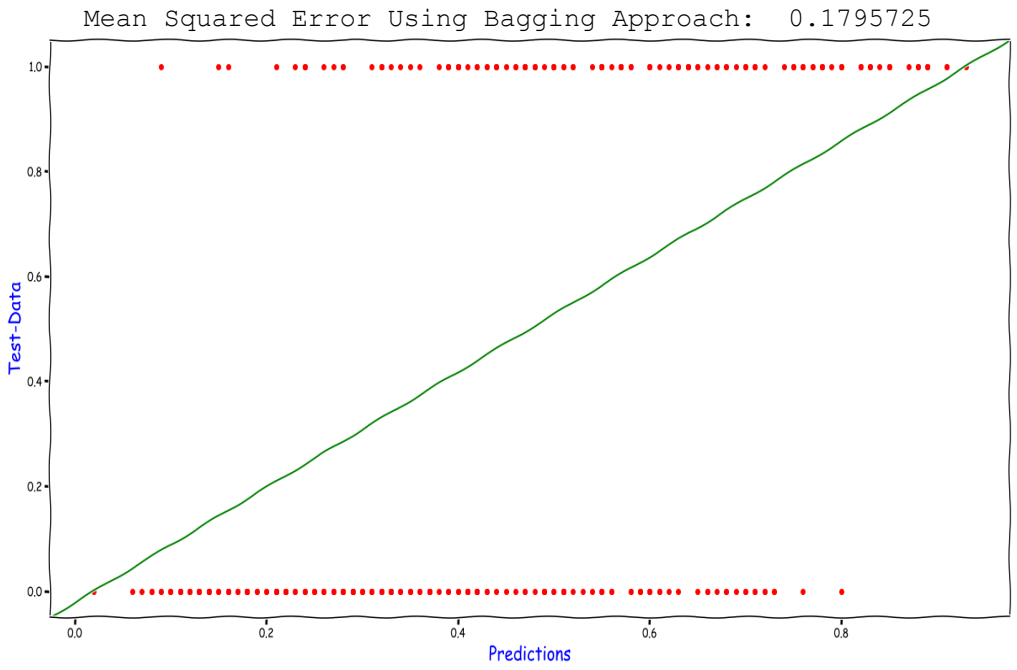

Fig4. MSE Using Bagging Approach

(e) Use random forests to analyse this data. What test MSE do you obtain?

Below code is performing random forest algorithm and calculating test MSE.

Random_Forest = RandomForestRegressor(max_features=3).fit(X_Train, Y_Train)

RF_prediction = Random_Forest.predict(X_Test)

plt.xkcd()

plt.figure(figsize=(24, 12))

plt.scatter(RF_prediction, Y_Test, label = 'medv', color='r')

plt.plot([0, 1], [0, 1], 'g', transform = plt.gca().transAxes)

plt.xlabel('Predictions',fontsize=22, color='b')

plt.ylabel('Test-Data',fontsize=22, color='b')

print("Mean Squared Error Using Random Forest Regressor: ", mean_squared_error(Y_Test, RF_prediction))

Output:

```
Mean Squared Error Using Random Forest Regressor:  0.1779821875
```

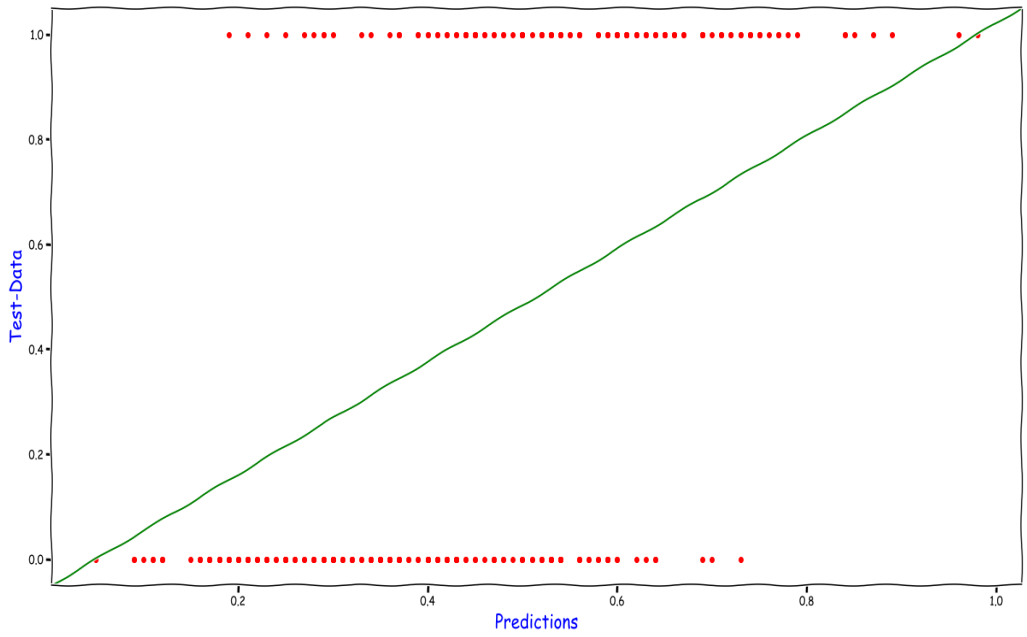

Fig5. MSE Using Random Forest Regressor

Question2. Perform K-Means clustering on Fish Market dataset (use download link from previous assignments) with K = 7 (No. of fish species). Normalize dataset before clustering. You can use Scikit Learn for clustering. How well do the clusters that you obtained in K-means clustering compare to the true class labels (Species)? Describe your results.

Here, we performing K-means clustering algorithm and before performing k means clustering, we normalize the data. We read csv file using pandas library. Below code is load our csv file.

```
Dataset1 = pd.read_csv('Fish.csv')

print(Dataset1)
```

Below code is normalize our data before applying K-means clustering.

```
from sklearn import preprocessing

X_normalization = preprocessing.normalize(X)
```

After normalized data we are implementing K-Means clustering algorithm and assigned label to each species. Here K=7 Species so each species have one cluster.

```
from sklearn.cluster import KMeans

kmeans_clustering = KMeans(n_clusters=7, random_state=0).fit(X_normalization)

kmeans_clustering.labels_
```

Output:

```
array([0, 0, 0, 0, 5, 5, 5, 0, 5, 5, 5, 5, 5, 0, 5, 5, 5, 5, 5, 5,
5, 5,5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 2, 2, 2, 2, 4, 3, 4, 4,
4,4, 4, 4, 4, 4, 4, 0, 4, 0, 0, 0, 0, 0, 0, 5, 5, 5, 2, 2, 4, 4, 4,
4, 0, 4, 0, 0, 0, 1, 2, 2, 2, 2, 4, 2, 2, 2, 2, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 0, 4, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5,
5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 4, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 5, 5, 5, 5, 5, 5, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 6, 1, 6, 6], dtype=int32)
```

K-means clustering algorithm calculate the distance from each centroid.

*kmeans_clustering.cluster_centers_*

kmeans_clustering.score(X_normalization)

Output:

Centroids for each cluster are:

```
Cluster 0:    Array
       ([[0.98422649, 0.09088637, 0.09824469, 0.10754415, 0.0291688 ,
        0.01478642],
Cluster 1:
       [0.41455842, 0.48455179, 0.51274857, 0.56146066, 0.09607448,
        0.0582441 ],
Cluster 2:
       [0.89453082, 0.22681553, 0.24809227, 0.27002538, 0.07411411,
        0.03894617],
Cluster 3:
       [0.        , 0.51630217, 0.55706286, 0.6195626 , 0.17595578,
        0.0910757 ],
Cluster 4:
       [0.96079027, 0.1425249 , 0.15539693, 0.1695393 , 0.04630177,
        0.0249035 ],
Cluster 5:
       [0.9957844 , 0.04605648, 0.04990506, 0.05561971, 0.018572  ,
        0.0082545 ],
Cluster 6:
       [0.58173996, 0.42981847, 0.46257423, 0.49842276, 0.09131453,
        0.05715133]])
```

Calculating score for K-means clustering using normalized data and using centroid of clusters.

from sklearn.metrics import silhouette_samples, silhouette_score

score = silhouette_score(X_normalization, kmeans_clustering.labels_, metric='euclidean')

print('Score:', score)

Output:

```
Score: 0.6104335904007592
```

Below code is assigned cluster label to each species.

*Dataset1['cluster'] = kmeans_clustering.labels_*

Dataset1.head()

Below code is Plot a graph for K-means Clustering graph for all seven species..

plt.figure(figsize=(12,7))

axis =
sns.barplot(x=np.arange(0,7,1),y=Dataset1.groupby(['cluster']).count()['Species'].values)

x=axis.set_xlabel("Cluster of Numbers")

x=axis.set_ylabel("Numbers")

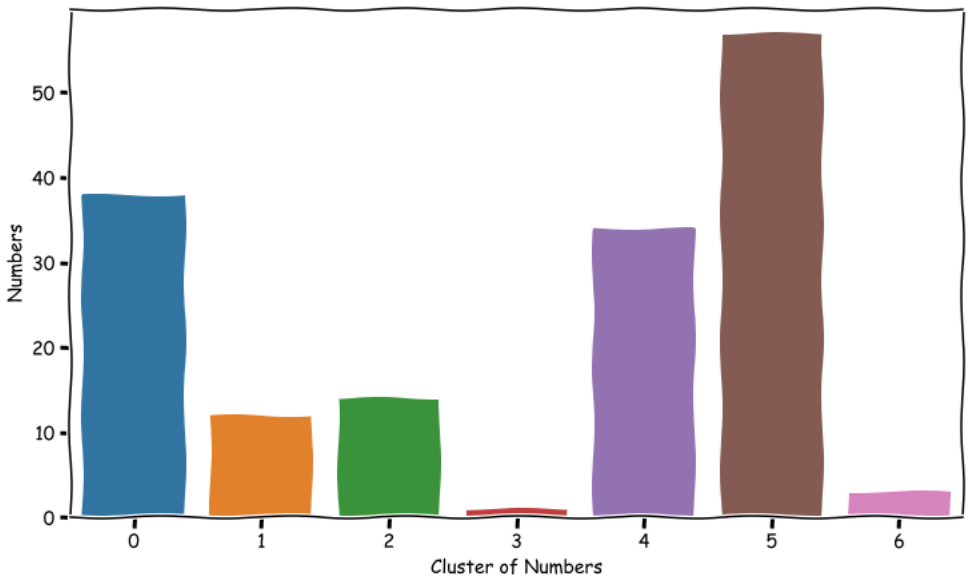

Fig6. Number of counts and cluster separations

Below code is plot graph for each cluster and number of species assigned to cluster.

sns.factorplot(col='cluster', y=None, x='Species', data=Dataset1, kind='count')

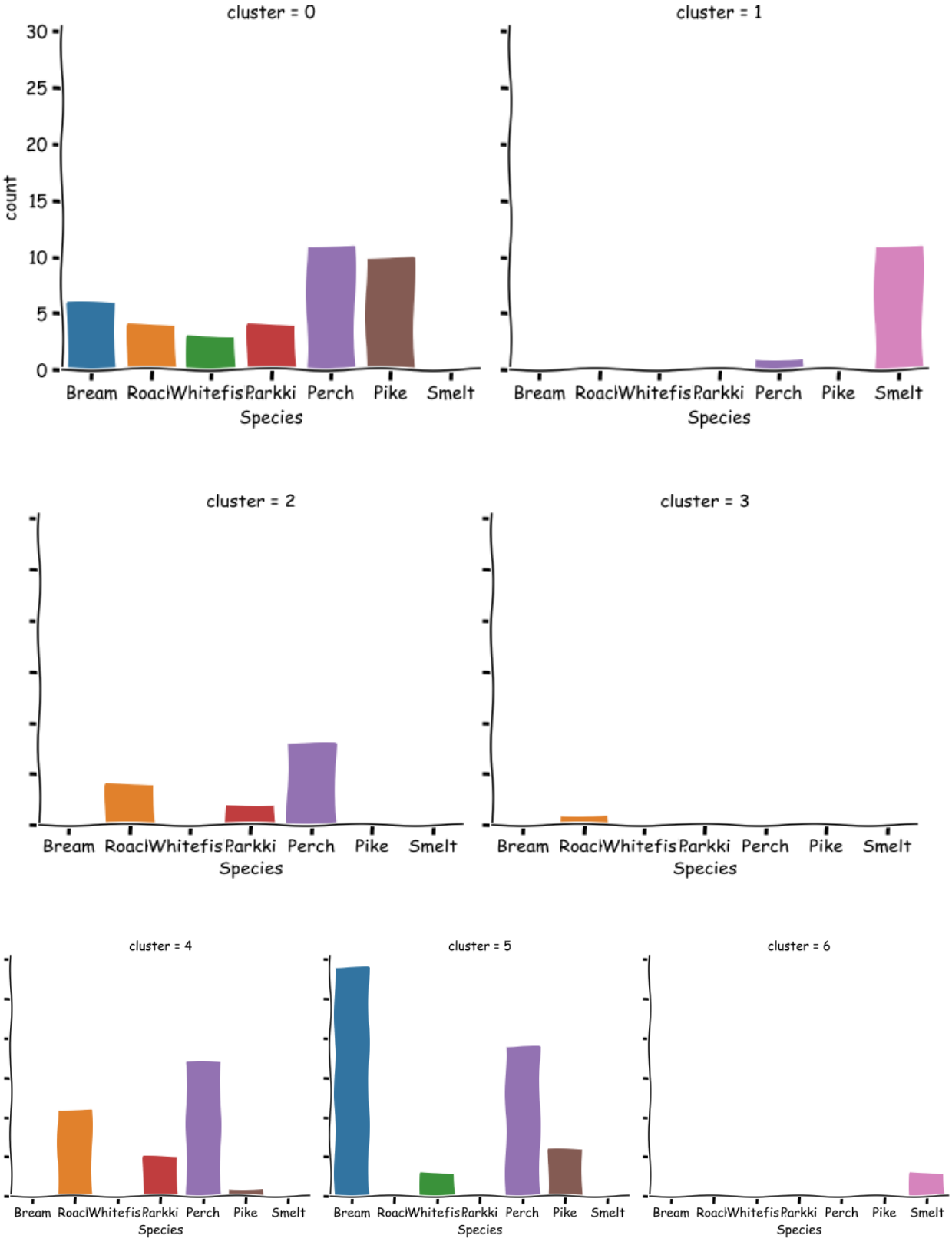

Fig. Graph for each cluster.

From the above graph we can say that cluster 3 and cluster 6 provides clear separation for the one individual species while other clusters have also some data points of another clusters. While the remaining cluster have the majority portion for the similarity species. Such as, in cluster 1 and cluster 5 have one species as majority. While in cluster 0,2 and 4 are not giving clear results for species. Overall, for this given data we are getting moderate clustering result in comparison of true label.