PREDICT THE AGE OF ABALONE BY CLASSIFICATION AND REGRESSION TREE

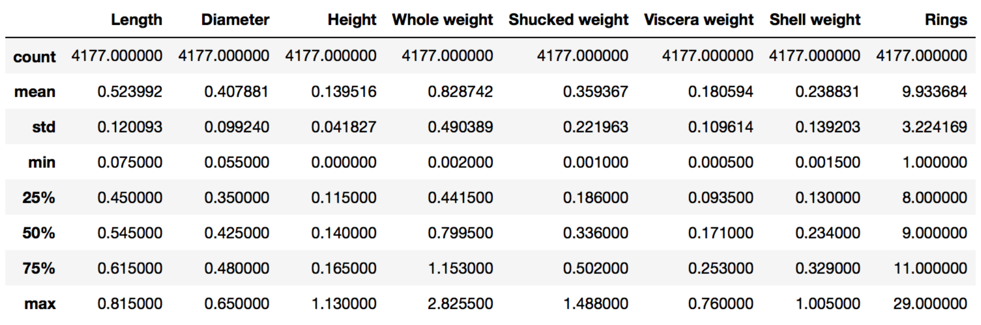
QINGHONG XU

Abstract. This project is based on the physical measurements, using the classification and regression tree to predict the age of abalone. Principal component analysis is implemented to select the important features in the dataset. Bootstrap aggregation and random forest are applied to improve the robustness and accuracy of the classification tree. We also compared the results using regression tree and lasso respectively.

Key words. classification and regression tree, bootstrap aggregation, random forest, abalone age, lasso

1. Introduction. Generally, the age of an abalone can be determined by cutting the shell and count the rings on it. To replace this time-consuming process, other measurements like the physical data for an abalone are used to predict the age. In this project, we used the dataset obtained from UCI machine learning repository, and applied the classification and regression tree (CART) to predict the age. Moreover, in order to improve the robustness and the accuracy of the prediction, we also used bootstrap aggregation and random forest to this dataset. This report will be organized as following: we first describe the features, the number of observations in this dataset; next, we preprocess the dataset and explore important properties of this dataset. In the third section, we investigate the correlation between different features and applied principal component analysis to reduce the dimension of the features. Then we fit the data with the classification tree model, bootstrap aggregation, random forest and perform the accuracy comparison.

2. Description of the dataset. Table 1 and Figure 1 summarize the information in this dataset. This dataset has 9 features: Sex, Length, Diameter, Height, Whole weight, Shucked weight, Viscera weight, Shell weight, Rings and 4177 observations in total. This table also indicates the 25, 50 and 75 percentile of different measures. For example, for the feature 'Length', below value 0.545 about 20% of the observations may be found.



	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings
count	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000	4177.000000
mean	0.523992	0.407881	0.139516	0.828742	0.359367	0.180594	0.238831	9.933684
std	0.120093	0.099240	0.041827	0.490389	0.221963	0.109614	0.139203	3.224169
min	0.075000	0.055000	0.000000	0.002000	0.001000	0.000500	0.001500	1.000000
25%	0.450000	0.350000	0.115000	0.441500	0.186000	0.093500	0.130000	8.000000
50%	0.545000	0.425000	0.140000	0.799500	0.336000	0.171000	0.234000	9.000000
75%	0.615000	0.480000	0.165000	1.153000	0.502000	0.253000	0.329000	11.000000
max	0.815000	0.650000	1.130000	2.825500	1.488000	0.760000	1.005000	29.000000

Fig. 1

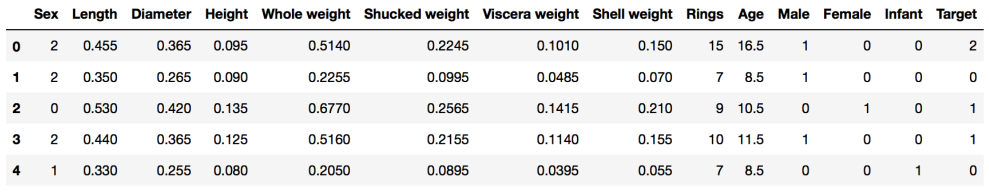
Feature	Data Type	Measurement Unit	Description
Sex	nominal	–	M, F, and I (infant)
Length	continuous	mm	Longest shell measurement
Diameter	continuous	mm	perpendicular to length
Height	continuous	mm	with meat in shell
Whole weight	continuous	grams	whole abalone
Shucked weight	continuous	grams	weight of meat
Viscera weight	continuous	grams	gut weight (after bleeding)
Shell weight	continuous	grams	after being dried
Rings	integer	–	1.5 gives the age in years

Table 1

3. Dataset preprocessing and exploration. From Figure 1 we could see the minimum value of feature ‘Highet’ is 0.000, which indicates that we missed the values of height for some observations. In order to deal with this situation, we found out those observations and deleted them in our dataset.

Also, as we can see, the data type of feature ‘Sex’ is nominal and there are three types: female, male and infant. There are two ways to encode the ‘Sex’: numeric encoding and one-hot encoder. Numeric encoding means we assign ‘Female’= 0.0, ‘Infant’= 1.0 , ‘Male’= 2.0. One-hot encoding means the features are encoded using a one-hot encoding scheme, which returns a binary column for each category. We can see Figure 2 for these two different encoding ways, and later we will compare their results.

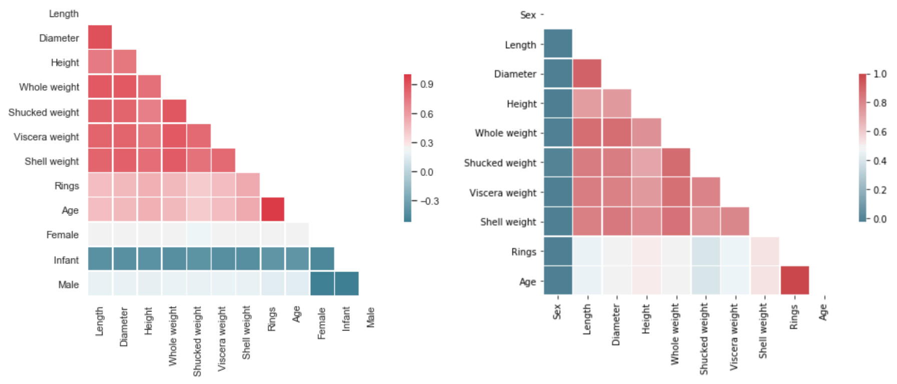
Besides, after adding 1.5 to the 'Rings' column, we obtained the 'Age' column. Originally there were 29 classifications, we reduced the categories to 3 categories to improve the classification rates, otherwise the accuracy rate would only float around 20%. In the end, we have three categories: 2.5 <= Age <= 9.5, encoded as 0 for the 'Young', 9.5 < Age <= 12.5 encoded as 1 for the 'Adult', and Age > 12.5 encoded as 2 for the 'Old'.



	Sex	Length	Diameter	Height	Whole weight	Shucked weight	Viscera weight	Shell weight	Rings	Age	Male	Female	Infant	Target
0	2	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15	16.5	1	0	0	2
1	2	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7	8.5	1	0	0	0
2	0	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9	10.5	0	1	0	1
3	2	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10	11.5	1	0	0	1
4	1	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7	8.5	0	0	1	0

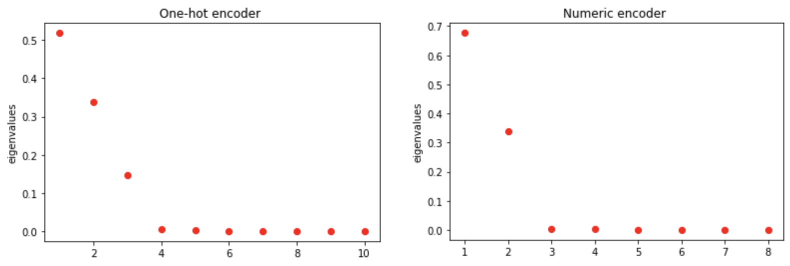
Fig. 2

4. The basic data analysis. After preprocessing the dataset, there were 13 features in total ('Sex' not included). We first want to explore the correlation between different features:



We use 'Kendall rank' to measure the correlation. Intuitively, two features will have high correlation if the observations have the similar rank in each feature. As we can see, for those physical features, such as 'Shell weight', 'Viscera weight', 'Shucked weight', 'Diameter', etc. they are highly related with each other, this seems reasonable. And this indicates that removing the highly-correlated features will not significantly affect the accuracy of our prediction.

Then we want to apply principal component analysis to reduce the dimension of features.



For one-hot encoder and numeric encoder, we all get a similar result, which is the first three components are important than the others. Therefore, we project the original dataset into the space spanned by the eigenvectors corresponding to the largest three eigenvalues. We apply PCA on the training set, then do the same transformation on the test set.

5. Classification and regression tree. Here we applied a simple but powerful method called classification and regression tree(CART) to classify our dataset. We restrict the attention to recursive binary partitions. We choose the feature and the variable and split the data into two regions. This process will continue on each node until the maximum depth of the tree or the minimum node records are achieved. Then the class for the terminal nodes or leaves will be the most common class in this region.

Suppose our data consists of p inputs and one target for each observation, that is $(x_i, y_i), i = 1, ..., N$ and $x_i = (x_{i1}, x_{i2}, ...x_{ip})$. Starting with all the data, we want to find a splitting feature j and value s to split our data into two regions:

$$R_1(j,s) = \{x|x_j \leq s\} \text{ and } R_2(j,s) = \{x|x_j > s\} \tag{5.1}$$

Now we need to decide the splitting feature and splitting value. For classification, we adopt our criterion minimization of the gini index and cross-entropy:

$$\text{Gini index} : \sum_{k \neq k'} \hat{p}_{mk}\hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{5.2}$$

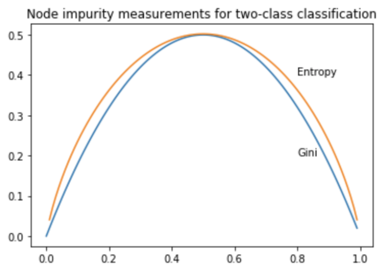
$$\text{Cross} - \text{entropy} : -\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk}) \tag{5.3}$$

where $\hat{p}_{mk}$ is the proportion of class k in node m. Denote the number of observations in node m by N_m, then we can define $\hat{p}_{mk}$ as:

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k) \tag{5.4}$$

Suppose we have two classes, then the gini index and the cross-entropy can be computed by $2p(1-p)$ and $-p\log p-(1-p)\log(1-p)$, where p denotes the proportion of the second class in one node.

As we can see from the following figure, Gini and entropy are useful to quantify the impurity of a node and differentiable for optimization needs. To calculate the node impurity for node m, we still need to weight them by the the number of observations in each child nodes (N_{mL}, N_{mR}) created by splitting the node m.



After defining the minimization criterion, we choose the feature and point that minimizes these two values and split recursively. We also determine the stopping criterion: the maximum depth of the tree and the minimum node records. At each terminal node m, we classify the observations in this node to be class $k(m) = \text{argmax}_k \hat{p}_{mk}$, the majority class in node m.

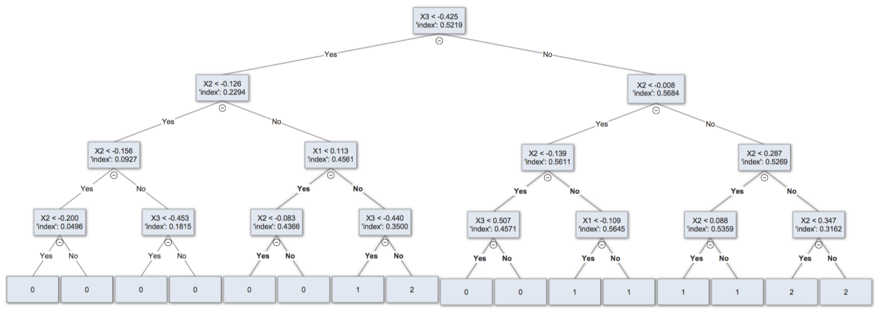

X3 < -0.425 'index': 0.5219
Yes
No
X2 < -0.126 'index': 0.2294
X2 < -0.008 'index': 0.5684
X2 < -0.156 'index': 0.0927
X1 < 0.113 'index': 0.4561
X2 < -0.139 'index': 0.5611
X2 < 0.287 'index': 0.5269
X2 < -0.200 'index': 0.0496
X3 < -0.453 'index': 0.1815
X2 < -0.083 'index': 0.4366
X3 < -0.440 'index': 0.3500
X3 < 0.507 'index': 0.4571
X1 < -0.109 'index': 0.5645
X2 < 0.088 'index': 0.5359
X2 < 0.347 'index': 0.3162
0
0
0
0
0
0
1
2
0
0
1
1
1
1
2
2

Fig. 3: A decision tree with maximum depth 4 minimum node records 2, gini index

In order to compare with the previous results, we used the same training and test set on the website, that is the final 1042 examples as test and the first 3133 used for training. The performance here is quantified by the misclassification error

	Training	Test
Gini, one-hot encoder	65.08%	63.05%
Cross entropy, one-hot encoder	65.36%	63.53%
Gini, numeric encoder	65.14%	61.61%
Cross entropy, numeric encoder	65.46%	61.22%

Table 2: The accuracy when using maximum depth 4 minimum node records 2

	Training	Test
Gini, one-hot encoder	76.60%	62.57%
Cross entropy, one-hot encoder	77.65%	64.29%
Gini, numeric encoder	77.88%	60.74%
Cross entropy, numeric encoder	76.85%	60.84%

Table 3: The accuracy when using maximum depth 9 minimum node records 2

As we can see, from the above two tables, the gini index and the cross entropy don't make much difference in the accuracy rate when applied in our data set. However, the data set with one-hot encoder outperformed than the data set with numeric encoder in the accuracy. This can be explained by that when using numeric encoder as 'Female'= 0.0, 'Infant'= 1.0 , 'Male'= 2.0, the 'Infant' class will never be classified. By comparison, the one-hot encoder doesn't have this issue and each category can be classified. Apart from that, as we increase the maximum depth of the tree, the training error decreased dramatically, while the test error didn't change a lot.

6. Bootstrap aggregation or bagging. Decision tree is a method with high variance, which means the accuracy of the prediction is sensitive to the training/test set. For example, when we used the first 1333 observations as test data and the last 3133 samples as training we obtained a different results, our test accuracy is 58.21% while the training accuracy is 67.38%. Therefore, we decided to use bootstrap aggregation to make CART more robust.

The data sample in a bootstrap is randomly drawn from our original dataset with replacements, this means one sample may be drawn more than once. The size of this bootstrap is the same as the original training set. This is done B times and we refit the data on each bootstrap. For each bootstrap, a decision tree is trained on this dataset, then the maximum output of these B bootstraps is used to make predictions.

We use cross validation to evaluate the performance of our model. We split the original dataset into 5 folds, 4 folds will be used as the training set and 1 fold will be used as test set. This means for each fixed number of B, we will fit the dataset on 5 models, then take the average as its performance. As we increase the bootstraps number from 1 to 5 and 10, the average accuracy improved. Note that the cross validation is applied on the whole data set.

Bootstraps #	1	2	3	4	5	Average
1	62.51%	63.11%	62.99%	64.91%	63.47%	63.40%
5	65.86%	65.02%	65.74%	63.71%	64.91%	65.054%
10	66.46%	65.38%	65.14%	64.31%	64.07%	65.07%

Table 4: Bootstrap aggregation, maximum depth 4 minimum node records 2, gini

7. Random forest. Although the bootstrap aggregation could reduce the high variance of the decision tree by generating multiple trees, it still suffered that the trees are highly correlated. This is because when performing the splitting by greedy algorithm it is very natural to have very similar split points, by comparing all the features and the values. In order to avoid this situation, we constraint those features which can be chosen, enforcing the splitting to be performed only under some random features. Suppose our data set has p features, then we would select $m < p$ of the input features at random as candidates for splitting. Typically values for m are $\sqrt{p}$. This is called random forest technique. For the following table, we chose $m = 2$. Notice that although the average accuracy improved with the increasing number of bootstraps, the accuracy is not better than no using random forest. This can be explained by here our $p = 3$, by choosing $m = 2$ is not making the features much more random and constrained.

Bootstraps #	1	2	3	4	5	Average
1	62.63%	62.27%	60.0%	63.95%	61.79%	62.132%
5	64.55%	64.19%	65.14%	65.02%	66.58%	65.10%
10	63.95%	67.42%	65.26%	64.43%	64.67%	65.15%

Table 5: random forest, maximum depth 4 minimum node records 2, gini

8. Regression tree and lasso regression. In this section, instead of regarding this problem as a classification problem, we treat this problem as a regression problem and use regression tree and lasso to predict the age of abalone. The difference between the regression tree and the classification tree is : At each terminal node m, we predict the observations in this node to be $k(m) = \frac{1}{N_m}\sum_{x_i \in R_m} y_i$, the average of the observations in node m. For lasso, we want to estimate the coefficients by

$$\hat{\beta} = \text{argmin}_\beta \sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 \text{ subject to } \sum_{j=1}^{p}|\beta_j| \leq t \tag{8.1}$$

Here we used the mean absolute error to describe the performance of these two algorithms.

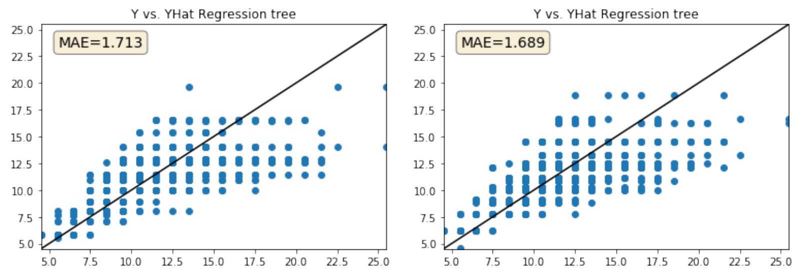


Fig. 4: Left: regression tree with dimension reduction; Right:regression tree without dimension reduction

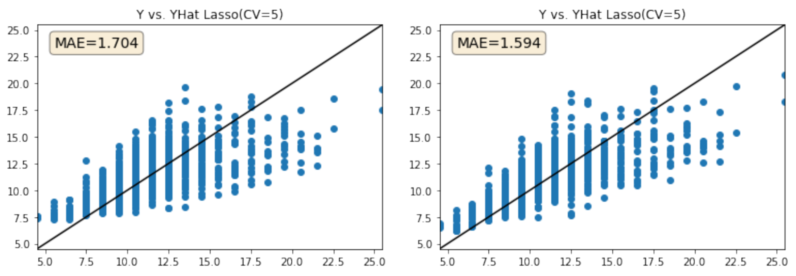


Fig. 5: Left: lasso with dimension reduction; Right: lasso without dimension reduction

Note that the lasso regression with cross validation 5 had a smaller mean absolute error compared with using regression tree, and in this case the dimension reduction didn't help reduce the error of prediction. Actually, for lasso we don't need to reduce the dimension, because of the nature of the constraint, making t sufficiently small will cause some of the coefficients to be exactly zero. Thus the lasso does a kind of continuous subset selection.

9. Conclusion and future work. From this project we learned that, we could see this prediction problem as either classification or regression problem. This depends on the type of information we need. If we want to know the age range of an abalone, such as young, adult or old. We can view this problem as a classification problem and used classification tree or linear/quadratic discriminant analysis to classify the age.

This project performed the principal component analysis first to reduce the feature dimensions. PCA is a dimension reduction technique that takes the variance of each feature into consideration, and infer new features that maximize the variance. Another dimension reduction technique called linear/quadratic discriminant analysis will also consider the class information and find a new feature subspace to best separate the data. Our future work will be considering quadratic discriminant analysis in this case and compare it with our results using PCA.

If we want to know the specific age of an abalone, regarding this problem as a regression problem may be a good way to do it. As we can see from previous section, both lasso and regression tree did produced similar results and dimension reduction didn't improve the mean absolute error in both algorithms.

From this project we also learned that reducing the dimension or features may not be a good choice for feature number is not large. The experiments in random forest and regression models implied no difference in reducing dimension. Though it may relive the storage and execution time issues, it didn't increase the performance of our model or solve the overfitting problem for our machine learning models. In the future, we will consider using dimensions reduction in the cases where the dataset contains large number of features or we would like a better performance of visualization(patterns, clusterings) in 2D or 3D.

REFERENCES

[1] Hastie, Friedman, Tibshirani, Friedman, J. H., & Tibshirani, Robert. (2009). The elements of statistical learning data mining, inference, and prediction / Trevor Hastie, Robert Tibshirani, Jerome Friedman. (2nd ed., Springer series in statistics). New York: Springer.

[2] https://machinelearningmastery.com/implement-random-forest-scratch-python/

[3] https://ericstrong.org/predicting-abalone-rings-part-1/

[4] https://ericstrong.org/predicting-abalone-rings-part-2/

[5] https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python/

MATH895 final project

December 6, 2018

```
In [ ]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as pl
        from sklearn import tree
        import pydot
        from sklearn.metrics import accuracy_score
        from sklearn.decomposition import PCA
        from sklearn import linear_model
        from random import seed
        from random import random
        from random import randrange
        from sklearn.tree import DecisionTreeRegressor
```

```
In [ ]: from sklearn.preprocessing import LabelEncoder, OneHotEncoder
        from sklearn.metrics import accuracy_score, mean_absolute_error
        from string import ascii_letters
        import seaborn as sns
        from numpy import ndarray
        from numpy import mean
        from numpy import cov
        from numpy.linalg import eig
        from sklearn.metrics import r2_score, mean_absolute_error
```

1 Data preprocessing

```
In [ ]: train_data = pd.read_csv("/Users/qinghongxu/Documents/MATH895/project/abalone.data.txt
                                 delimiter=',' )
```

```
In [ ]: train_data = train_data[train_data['Height']>0]
```

```
In [ ]: train_data['Age'] = train_data['Rings'] + 1.5
```

```
In [ ]: v = train_data['Age'].values.tolist()
        a = [0] * len(v)
        for i in range(len(v)):
            if v[i] <= 9.5:
                a[i] = 0
            elif v[i] >12.5:
                a[i] = 2
            else:
                a[i] = 1
```

```
In [ ]: # 3 - category classfication: 2.5<=Age<=9.5, 9.5<Age<=12.5, Age>12.5
        train_data['Target'] = a
```

```
In [ ]: train_data['Male'] = (train_data['Sex']=='M').astype(int)
        train_data['Female'] = (train_data['Sex']=='F').astype(int)
        train_data['Infant'] = (train_data['Sex']=='I').astype(int)
```

```
In [ ]: train_data['Sex'] = LabelEncoder().fit_transform(train_data['Sex'].tolist()) #F=0.0, I
```

# Basic Data Analysis

```
In [ ]: #train_data = train_data.drop('Target', axis = 1)
        train_data = train_data.drop('Sex', axis = 1)
        #train_data = train_data.drop('Rings', axis = 1)
        corr = train_data.corr('kendall')
        mask = np.zeros_like(corr, dtype=np.bool)
        mask[np.triu_indices_from(mask)] = True
        f, ax = pl.subplots(figsize=(8, 6))
        cmap = sns.diverging_palette(220, 10, as_cmap=True)
        sns.heatmap(corr, mask=mask, cmap=cmap, vmax=1,square=True,
                    linewidths=.5, cbar_kws={"shrink": .5}, ax=ax);
```

```
In [ ]: Ytrain_data = train_data['Target']
        #train_data = train_data.drop('Sex',axis = 1)
        train_data = train_data.drop('Rings',axis = 1)
        train_data = train_data.drop('Age',axis = 1)
        train_data = train_data.drop('Target',axis = 1)
```

```
In [ ]: train_data = train_data.drop('Sex',axis = 1)
```

```
In [ ]: # PCA
        A = train_data[0:3133].values
        M = mean(A.T, axis=1)
        C = A - M
        V = cov(C.T)
        values, vectors = eig(V)
        P = vectors[0:3].dot(C.T)
        P.T.shape
```

```
In [ ]: train_data = train_data.drop('Sex',axis = 1)
```

```
In [ ]: Ytrain_data = train_data['Age']
```

```
In [ ]: train_data = train_data.drop('Rings',axis = 1)
        train_data = train_data.drop('Age',axis = 1)
```

2 Decision tree

```
In [ ]: def gini_index(groups, classes):
            all_samples = float(sum(len(group) for group in groups))
            gini = 0.0
            for group in groups:
                size = float(len(group))
                probability = 0.0
                if size == 0:
                    continue
                for class_index in classes:
                    p = [row[-1] for row in group].count(class_index) / size
                    probability += p * p
                gini += (1 - probability)* size/all_samples
            return gini
```

```
In [ ]: def entropy_index(groups, classes):
            all_samples = float(sum(len(group) for group in groups))
            entropy = 0.0
            for group in groups:
                size = float(len(group))
                probability = 0.0
                if size == 0:
                    continue
                for class_index in classes:
                    p = [row[-1] for row in group].count(class_index) / size
                    if p != 0:
                        probability += p * np.log(p)
                entropy -=  probability * size/all_samples
            return entropy
```

```
In [ ]: def test_split(index, value, dataset):
            left, right = list(), list()
            for row in dataset:
                if row[index] < value:
                    left.append(row)
                else:
                    right.append(row)
            return left, right
```

```
In [ ]: def get_split(dataset, measurement):
            b_index, b_value, b_score, b_groups = 999, 999, 999, None
            class_values = list(set(row[-1] for row in dataset))
            features = list()
            while len(features)<3:
                index = randrange(len(dataset[0])-1)
                if index not in features:
                    features.append(index)
            for index in features:
```

```
            for row in dataset:
                groups = test_split(index, row[index], dataset)
                if measurement == 1:
                    score = gini_index(groups, class_values)
                    #print('X%d < %.3f Gini=%.3f' % ((index+1), row[index], score))
                else:
                    score = entropy_index(groups, class_values)
                    #print('X%d < %.3f Entropy=%.3f' % ((index+1), row[index], score))
                if score < b_score:
                         b_score, b_index, b_value, b_groups = score, index, row[index], g
        return {'gini':b_score, 'index':b_index, 'value':b_value, 'groups':b_groups}
```

```
In [ ]: def terminal_outcome(node):
            outcomes = [row[-1] for row in node]
            return max(set(outcomes), key = outcomes.count)
```

```
In [ ]: def split(max_depth, min_size, node, depth, measurement):
            left, right = node['groups']
            del node['groups']
            if not left or not right:
                node['left'] = node['right'] =  terminal_outcome(right + left)
                return
            if depth >= max_depth:
                node['left'], node['right'] = terminal_outcome(left), terminal_outcome(right)
                return
            if len(left) <= min_size:
                node['left'] = terminal_outcome(left)
            else:
                node['left'] = get_split(left, measurement)
                split(max_depth, min_size, node['left'], depth+1, measurement)
            if len(right) <= min_size:
                node['right'] = terminal_outcome(right)
            else:
                node['right'] = get_split(right, measurement)
                split(max_depth, min_size, node['right'], depth+1, measurement)
```

```
In [ ]: def build_tree(max_depth, min_size, measurement, dataset):
            root = get_split(dataset,measurement)
            split(max_depth, min_size, root, 1, measurement)
            return root
```

```
In [ ]: def print_tree(node, depth = 0):
            if isinstance(node, dict):
                print('%sX%d < %.3f' % (depth*' ', node['index']+1, node['value']))
                print_tree(node['left'], depth+1)
                print_tree(node['right'], depth+1)
            else:
                print('%s%d' % (depth*' ', node))
```

```
In [ ]: def predict(node, row):
            if row[node['index']] < node['value']:
                if isinstance(node['left'], dict):
                    return predict(node['left'], row)
                else:
                    return node['left']
            else:
                if isinstance(node['right'], dict):
                    return predict(node['right'], row)
                else:
                    return node['right']
```

```
In [ ]: def prediction(tree,test_data):
            predictions = list()
            for row in test_data:
                prediction = predict(tree, row)
                predictions.append(prediction)
            return predictions
```

```
In [ ]: def accuracy(actual, predict):
            correct = 0.0
            for row in range(len(predict)):
                if actual[row] == predict[row]:
                    correct += 1
            return float(correct)/len(predict) * 100
```

```
In [ ]: def decision_tree(train_data,test_data, max_depth, min_size, measurement):
            actual_train = list()
            actual_test = list()
            tree = build_tree(max_depth, min_size, measurement,train_data)
            print_tree(tree)
            predictions_train = prediction(tree,train_data)
            predictions_test = prediction(tree,test_data)
            for row in train_data:
                actual_train.append(row[-1])
            for row in test_data:
                actual_test.append(row[-1])
            accuracy_train = accuracy(actual_train, predictions_train)
            accuracy_test = accuracy(actual_test, predictions_test)
            return accuracy_train, accuracy_test, tree
```

```
In [ ]: Xtrain = P.T
        C = train_data[3133:4177] - M
        P = vectors[0:3].dot(C.T)
        Xtest = P.T
        Ytrain = Ytrain_data[0:3133]
        Ytest = Ytrain_data[3133:4177]
```

```
In [ ]: len(Xtrain)
In [ ]: Ytrain = Ytrain.values
        Xtrain_my = np.column_stack((Xtrain, Ytrain))
        Ytest = Ytest.values
        Xtest_my = np.column_stack((Xtest, Ytest))
```

```
In [ ]: accuracy_train, accuracy_test, tree = decision_tree(Xtrain_my, Xtest_my, 4, 2, 1)
```

```
In [ ]: p = np.arange(0., 1., 0.01)
        y = 2*p*(1-p)
        yy = (-p*np.log(p)-(1-p)*np.log(1-p))*0.7246
        pl.plot(p,y,p,yy)
        pl.text(0.8, 0.2, 'Gini')
        pl.text(0.8, 0.4, 'Entropy')
        pl.title('Node impurity measurements for two-class classification')
        pl.show()
```

```
In [ ]: def subsample(dataset, ratio):
            sample = list()
            total = round(ratio * len(dataset))
            while len(sample) < total:
                index = randrange(len(dataset))
                sample.append(dataset[index])
            return sample
```

```
In [ ]: def bagging_predict(trees, row):
            predictions = [predict(tree, row) for tree in trees]
            return max(set(predictions), key = predictions.count)
```

```
In [ ]: def bagged_tree(train, test, ntrees, max_depth, min_size, measurement):
            trees = list()
            for n in range(ntrees):
                sample = subsample(train, 1)
                tree = build_tree(max_depth, min_size, measurement, sample)
                trees.append(tree)
            predictions = [bagging_predict(trees, row) for row in test]
            return predictions
```

```
In [ ]: def cross_validation_split(dataset, n_folds):
            dataset_split = list()
            dataset_copy = list(dataset)
            size = int(len(dataset)/n_folds)
            for i in range(n_folds):
                fold = list()
                while len(fold) <  size:
                    index = randrange(len(dataset_copy))
                    fold.append(dataset_copy.pop(index))
                dataset_split.append(fold)
            return dataset_split
```

```
In [ ]: def evaluate_cv(dataset, n_folds, *args):
            folds = cross_validation_split(dataset, n_folds)
            scores = list()
            for fold in folds:
                train_set = list(folds)
                train_set.remove(fold)
                train_set = sum(train_set,[])
                test_set = list()
                for row in fold:
                    row_copy = list(row)
                    #row_copy[-1] = None
                    test_set.append(row_copy)
                actual = [row[-1] for row in fold]
                predictions = bagged_tree(train_set, test_set, *args)
                accuracy_test = accuracy(actual, predictions)
                scores.append(accuracy_test)
            return scores
```

```
In [ ]: def evaluate(trainset, testset, *args):
            train_set = list()
            test_set = list()
            scores = list()
            for row in trainset:
                row_copy = list(row)
                #row_copy[-1] = None
                train_set.append(row_copy)
            for row in testset:
                row_copy = list(row)
                #row_copy[-1] = None
                test_set.append(row_copy)
            actual = [row[-1] for row in testset]
            predictions = bagged_tree(train_set, test_set, *args)
            accuracy_test = accuracy(actual, predictions)
                #scores.append(accuracy_test)
            return accuracy_test
```

```
In [ ]: train_data_my = np.column_stack((train_data.values, Ytrain_data.values))
```

```
In [ ]: for n in [1,5,10]:
        scores = evaluate(Xtrain_my.tolist(), Xtest_my.tolist(), n, 4,2,1)
        print('Accuracy: %f' % scores)
            #print('Mean Accuracy: %.3f%%' % (sum(scores)/float(len(scores))))
```

3 Regression

```
In [ ]: train_data = train_data.drop('Sex', axis = 1)
        Ytrain_data = train_data['Age'][0:3133]
        Ytest_data = train_data['Age'][3133:4177]
        train_data = train_data.drop('Age', axis = 1)
        train_data = train_data.drop('Rings', axis = 1)
        Xtrain_data = train_data[0:3133]
        Xtest_data = train_data[3133:4177]
```

```
In [ ]: len(Xtest)
```

```
In [ ]: regr = DecisionTreeRegressor(max_depth=4)
        regr.fit(Xtrain, Ytrain)
        predict = regr.predict(Xtest)
```

```
In [ ]: absmin = min([Ytest.min(),predict.min()])
        absmax = max([Ytest.max(),predict.max()])
        ax = pl.axes()
        ax.scatter(Ytest,predict)
        ax.set_title('Y vs. YHat Regression tree')
        ax.axis([absmin, absmax, absmin, absmax])
        ax.plot([absmin, absmax], [absmin, absmax],c="k")
        mae = mean_absolute_error(Ytest_data, predict)
        props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
        textStr = 'MAE=%.3f' % mae
        ax.text(0.05, 0.95, textStr, transform=ax.transAxes, fontsize=14, verticalalignment='to
```

```
In [ ]: lasso_model = linear_model.LassoCV(cv=5).fit(Xtrain, Ytrain)
        ypred2 = lasso_model.predict(Xtest)
```

```
In [ ]: absmin = min([Ytest.min(),ypred2.min()])
        absmax = max([Ytest.max(),ypred2.max()])
        ax = pl.axes()
        ax.scatter(Ytest_data,ypred2)
        ax.set_title('Y vs. YHat Lasso(CV=5)')
        ax.axis([absmin, absmax, absmin, absmax])
        ax.plot([absmin, absmax], [absmin, absmax],c="k")
        mae = mean_absolute_error(Ytest_data, ypred2)
        props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)
        textStr = 'MAE=%.3f' % mae
        ax.text(0.05, 0.95, textStr, transform=ax.transAxes, fontsize=14, verticalalignment='to
```