Introduction to Machine Learning Program assignment #3

Environment: Ubuntu 16.04.3 LTS

Language: Python 2.7.12

Using library:

```
import pandas as pd
import numpy as np
import sklearn
from sklearn import tree
from sklearn.cross_validation import train_test_split

from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score

from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from sklearn.naive_bayes import MultinomialNB
from sklearn.preprocessing import normalize, MinMaxScaler
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

Dataset1: Iris.csv

Data information:

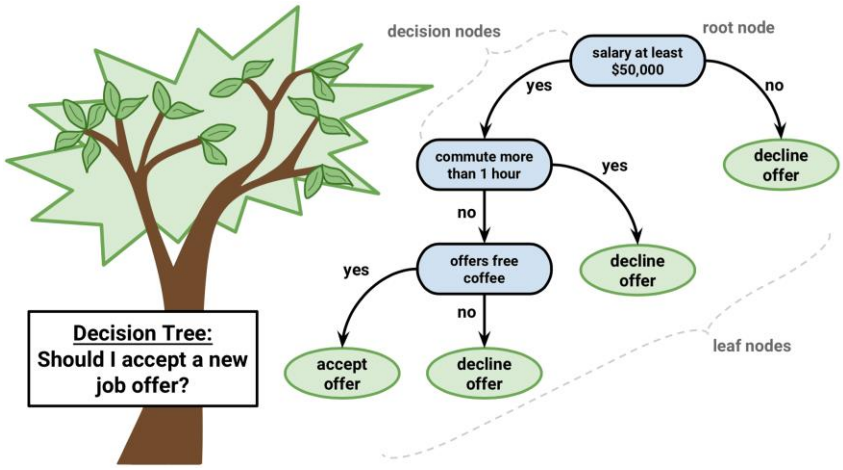
features: the sepals length/width and petals length/width

```
df.describe()
```

	0	1	2	3
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Method1: Decision Tree

1. Algorithm:



(references:http://dataaspirant.com/2017/01/30/how-decision-tree-algorithm-works/)
At the beginning, the whole training set is considered as the root.
Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
Records are distributed recursively on the basis of attribute values.
Order to placing attributes as root or internal node of the tree is done by using calculating entropy.

2. Training Ways:

After spilting datas into training set(70%) and testing set(30%) ,

```
x_train, x_test, y_train, y_test = train_test_split(df_data, df_target, test_size=0.3)
```

we use sklearn package to define the decision tree classfier

```
clf = tree.DecisionTreeClassifier()
```

.

Then put training data and training target into the classifier to fit model

```
clf = clf.fit(x_train, y_train)
```

, here we got the trained model, we can start to test.

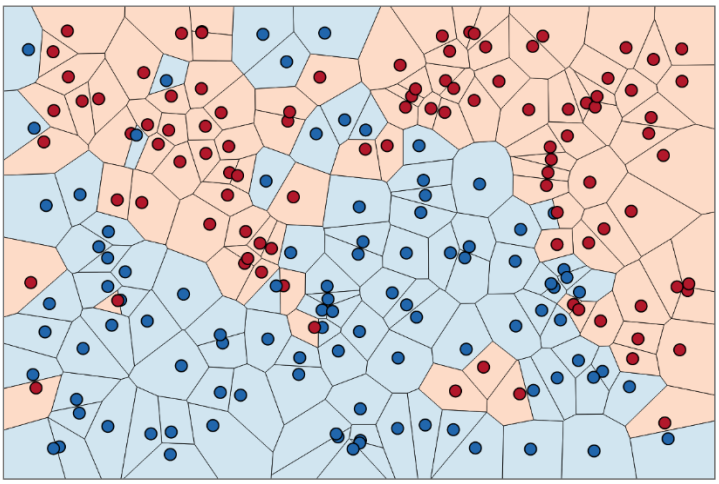
3. Result:

I calculate three types of values to evualuate the performance of my Decision tree.
First of all, score (X, y, sample_weight=None) will returns the **mean accuracy** on the given test data and labels.
Besides, **recall_score**(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None) will compute the recall.
Finally, sklearn.metrics.**precision_score**(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None) computes the precision.
It turns out that:

```
mean accuracy: 0.955555555556
recall: 0.962962962963
precision: 0.964912280702
```

Method2: K-nearest beighbor(with n=3)

1. Algorithm:



(references: https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/#how-does-knn-work)

In the classification setting, the K-nearest neighbor algorithm essentially choose is the Euclidean distance given by $d(x,x') = \sqrt{(x_1 - x'_1)^2 + (x_2 - x'_2)^2 + \ldots + (x_n - x'_n)^2}$ or other distance formula such as Manhattan, Chebyshev and Hamming distance.

2. Training Ways:

   Use sklearn package to define the KNeighborsClassifier `neigh = KNeighborsClassifier(n_neighbors=3)`. Put training data and training target into the classifier to fit model `neigh.fit(x_train, y_train)`, here we got the trained model, we can start to test.

3. Result:

```
mean accuracy: 0.977777777778
recall: 0.981481481481
precision: 0.981481481481
```

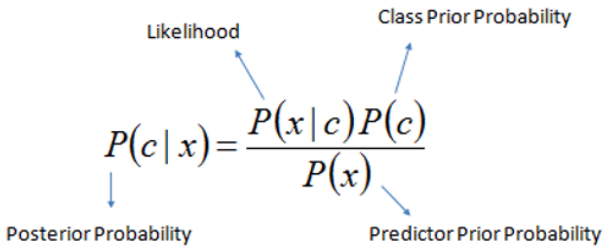
4. Pros & Cons:

   Pros: Simple to implement, Flexible to feature/distance choices, Naturally handles multi-class cases, Can do well in practice with enough representative data

   Cons: Large search problem to find nearest neighbours, Storage of data, Must know we have a meaningful distance function

Method3: Naive Bayes

1. Algorithm:



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

(references: http://dataaspirant.com/2017/02/06/naive-bayes-classifier-machine-learning/)

Naive Bayes works on conditional probability. Conditional probability is the probability that something will happen, given that something else has already occurred. Using the **conditional probability**, we can calculate the probability of an event using its prior knowledge.

2. Training Ways:

   Because Iris dataset uses **continuous features**, we **apply Gaussian smooth** to the classifier. Use sklearn package to define the GaussianNB Classifier `gnb = GaussianNB()`. Put training data and training target into the classifier to fit model `gnb.fit(x_train, y_train)`, here we got the

trained model, we can start to test.

3. Result:

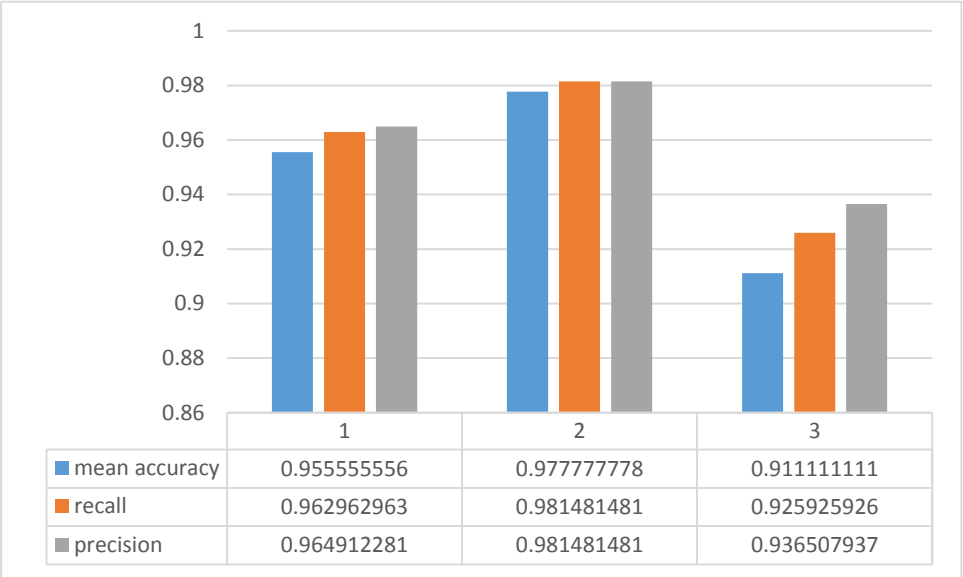
```
mean accuracy: 0.911111111111
recall: 0.925925925926
precision: 0.936507936508
```

4. Pros & Cons:

Pros: Computationally fast, Simple to implement, Works well with high dimensions

Cons: Relies on independence assumption and will perform badly if this assumption is not met

Comparison:



	1	2	3
mean accuracy	0.955555556	0.977777778	0.911111111
recall	0.962962963	0.981481481	0.925925926
precision	0.964912281	0.981481481	0.936507937

(1: Decision Tree; 2: K-neares; 3: Naive Bayes)

Performance: KNN > Decision Tree > Naive Bayes

Decision Trees is easy to interpret visually when the trees only contain several levels, it can easily handle qualitative (categorical) features. Morever, it works well with decision boundaries parellel to the feature axis! So it behave well when classifying Iris dataset. However, naive bayes relies on independence assumption, so it may perform badly if this assumption is not met.

Dataset2: forestfires.csv

Data information:

1.

```
df.head()
```

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	7	5	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0
3	8	6	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.0
4	8	6	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.0

Features ‘month’ and ‘day’ are still be recorded by string, so we should **convert it to int type** to do the classfication.

Replacing month from「January~December」 with 「1~12」; day from「Sunday~Saturday」 with 「1~7」, after doing such replacement, dataframe becomes:

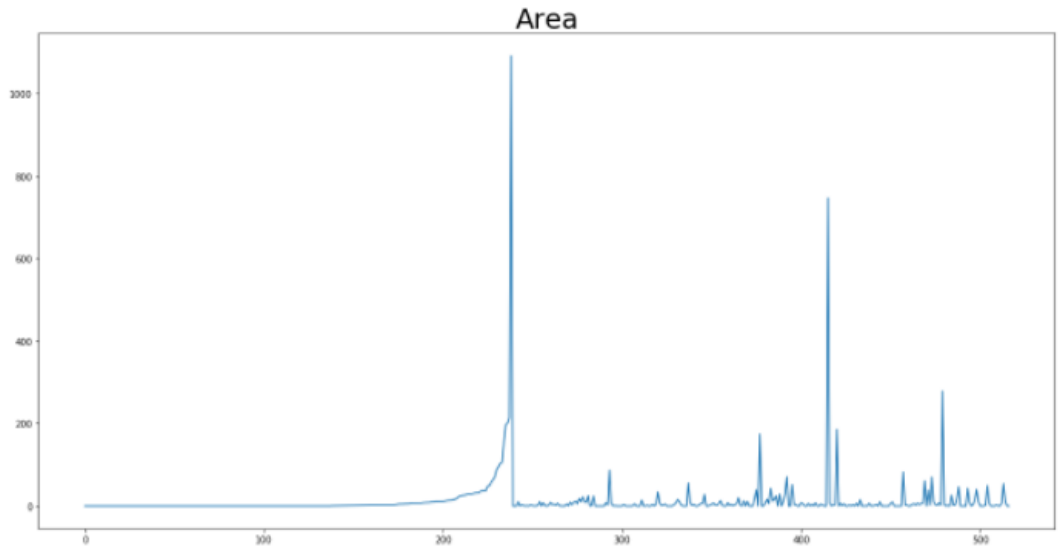
```
df.head()
```

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	7	5	3	5	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0
1	7	4	10	2	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0
2	7	4	10	6	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0
3	8	6	3	5	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.0
4	8	6	3	7	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.0

2.

```
plt.figure(figsize=(20,10))
plt.title('Area',fontsize=30)
plt.plot(df['area'])
```

```
[<matplotlib.lines.Line2D at 0x7f8fca34b2d0>]
```

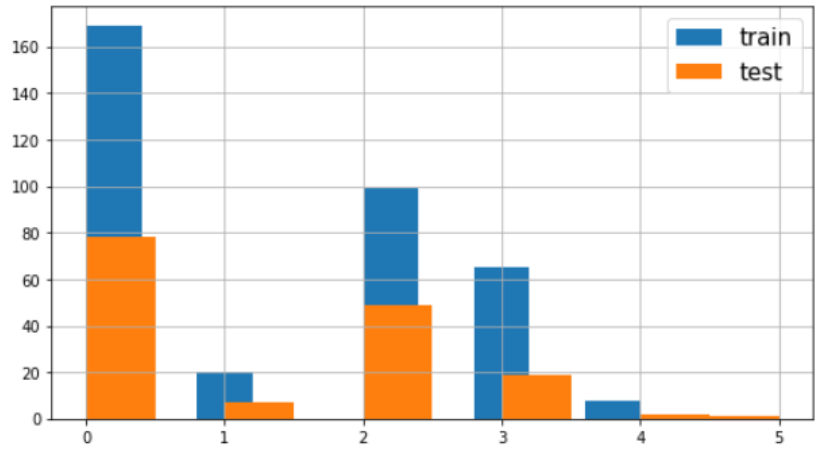


Plot the target(‘area’), we found out that the **curve is skewed**, amd it’s tough for naive bayes to

perdict **continuous target**, so in Method 3(Naive Bayes), we **divided target to 6 class**(0, 0-1, 1-10, 10-100, 100-1000, >1000). The new area class is distributes as belowed:

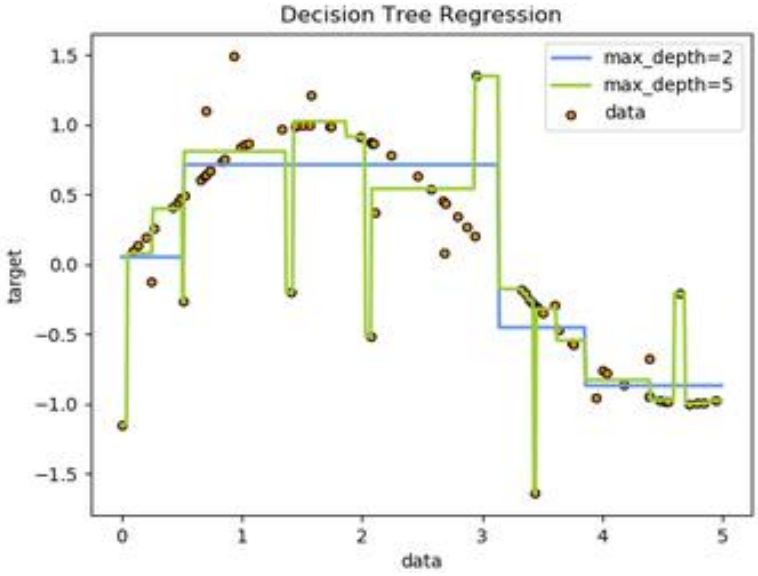
```
def area_to_class(y):
    if y==0:
        return 0
    elif y>0 and y<=1:
        return 1
    elif y>1 and y<=10:
        return 2
    elif y>10 and y<=100:
        return 3
    elif y>100 and y<=1000:
        return 4
    elif y>1000:
        return 5

y_train_class = y_train.apply(area_to_class)
y_test_class = y_test.apply(area_to_class)
y_train_class.hist(figsize = (9,5), label='train')
y_test_class.hist(figsize = (9,5), label = 'test')
plt.legend(fontsize = 15)
```

```
<matplotlib.legend.Legend at 0x7f8fca88bf90>
```



Method1: Decision Tree Regressor

1. Algorithm:



(references: http://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html)

A 1D regression with decision tree.

The decision trees is used to fit a sine curve with addition noisy observation. As a result, it learns local linear regressions approximating the sine curve.

We can see that if the maximum depth of the tree (controlled by the max_depth parameter) is set too high, the decision trees learn too fine details of the training data and learn from the noise, i.e. they overfit.

2. Training Ways:

After spilting datas into training set(70%) and testing set(30%) ,

```
x_train, x_test, y_train, y_test = train_test_split(df_data, df_target, test_size=0.3)
```

we use sklearn package to define the decision tree regressor

```
regressor = DecisionTreeRegressor(random_state=0)
```

.

Then put training data and training target into the regressor to fit model

```
regressor.fit(x_train,y_train)
```

, here we got the trained model, we can start to test.

3. Result:

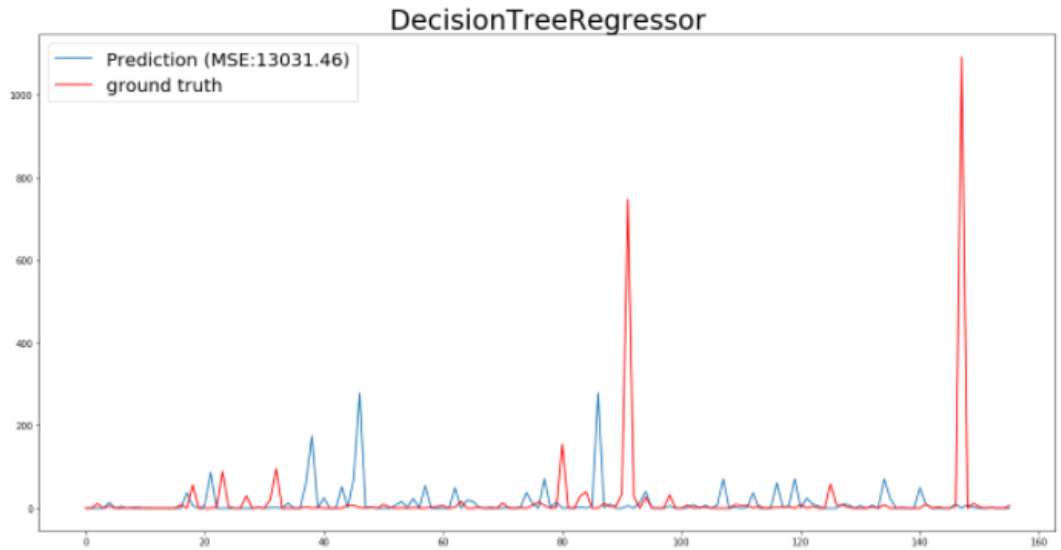
To evaluate the result of regression, it's crucial to have mean square error for the continuous target. Therefore, I calculate the **mean square error** and the **mean accuracy**, getting the results below:

```
mean square error:  13031.4637333
mean accuracy: -0.159926936746
```

In addition, I plot the curve of testing result together with the ground truth of area:

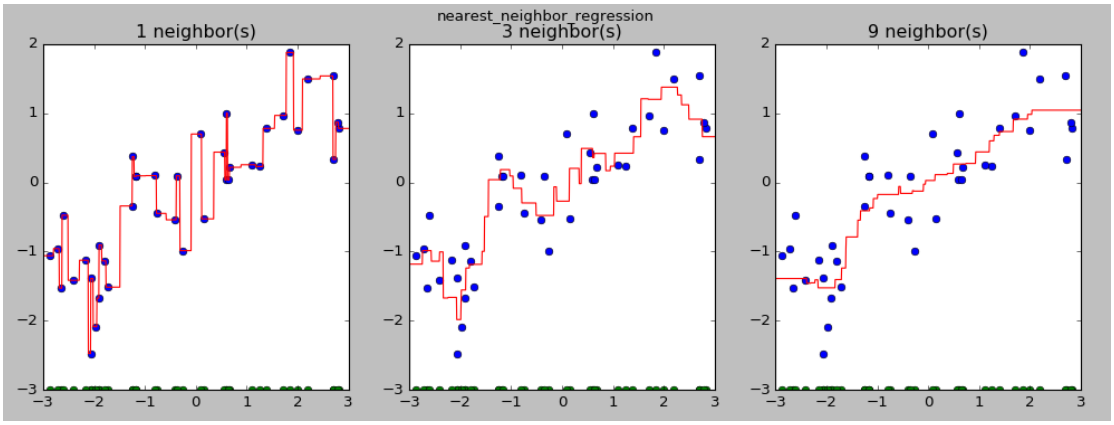
```
plt.figure(figsize=(20,10))
plt.title('DecisionTreeRegressor',fontsize=30)
plt.plot(index,regressor.predict(x_test),label="Prediction (MSE:%0.2f)" %mse)
plt.plot(index,y_test,c='r',label="ground truth")
plt.legend(fontsize = 20)
print 'mean accuracy:',regressor.score(x_test,y_test)
```

```
mean square error:  13031.4637333
mean accuracy: -0.159926936746
```



Method2: K-nearest beighbor Regressor

1. Algorithm:



(references: http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html)

Regression based on k-nearest neighbors. The target is predicted by local interpolation of the targets associated of the nearest neighbors in the training set.

2. Training Ways:

Use sklearn package to define the k-nearest neighbors regressor

```
neigh = KNeighborsRegressor(n_neighbors=3)
```

.

Then put training data and training target into the regressor to fit model

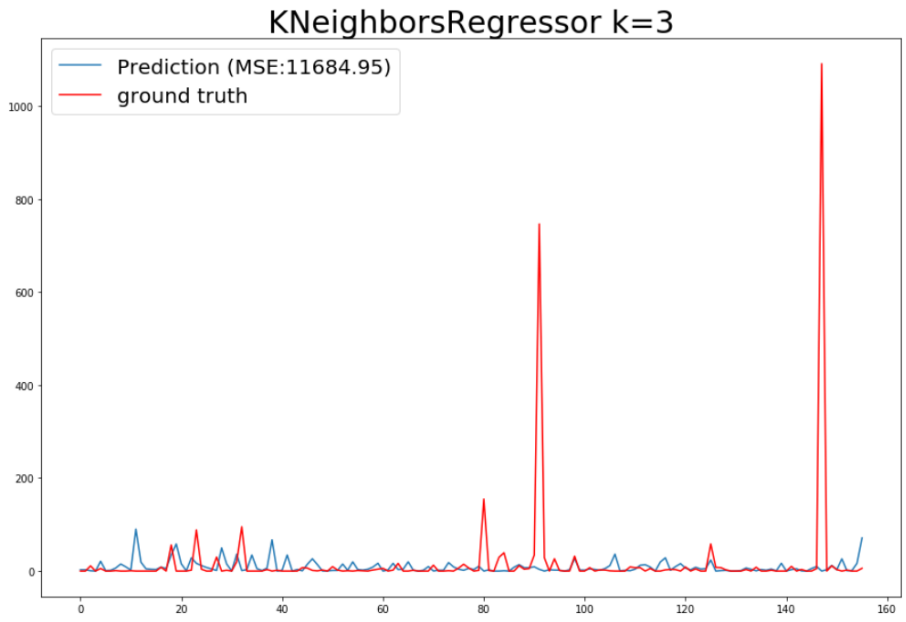
```
neigh.fit(x_train,y_train)
```

, here we got the trained model, we can start to test.

3. Result:

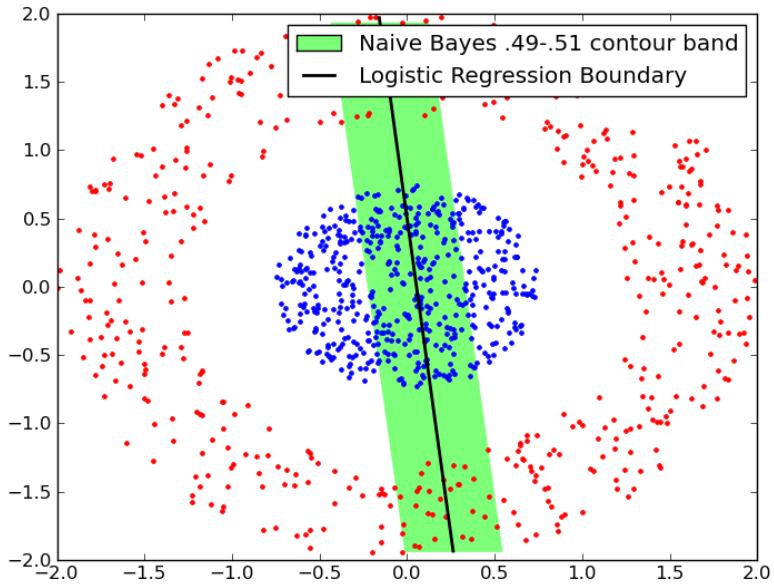
```
mean square error : -0.04007385623
mean square error:  11684.9469635
```

Predicted area VS. Ground truth:



Method3: Naive Bayes Regressor

1. Algorithm:



(references: http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html)

Gaussian Naive Bayes (GaussianNB) performs continuous features' updates to model parameters via partial_fit method; while multinomial Naive Bayes classifier is suitable for classification with discrete features (e.g., word counts for text classification). The multinomial distribution normally requires integer feature counts.

2. Training Ways:

Because data contains both continuous and discrete features ,we use two different models: MultinomialNB for discrete features such as 'X', 'Y', 'month', 'day' 、GaussianNB for continuous features such as 'FFMC', 'DMC', 'ISI', 'temp' and so on.

(1) MultinomialNB

$$p(\mathbf{x} \mid C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i p_{ki}{}^{x_i}$$

The multinomial naive Bayes classifier becomes a linear classifier when expressed in log-space:

$$\begin{aligned} \log p(C_k \mid \mathbf{x}) &\propto \log\left(p(C_k) \prod_{i=1}^{n} p_{ki}{}^{x_i}\right) \\ &= \log p(C_k) + \sum_{i=1}^{n} x_i \cdot \log p_{ki} \\ &= b + \mathbf{w}_k^\top \mathbf{x} \end{aligned}$$

where $b = \log p(C_k)$ and $w_{ki} = \log p_{ki}$.

(2) GaussianNB

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

Use sklearn package to define the decision tree regressor
`neigh = KNeighborsRegressor(n_neighbors=3)` .

Then put training data and training target into the regressor to fit model
`neigh.fit(x_train,y_train)` , here we got the trained model, we can start to test.

(3) Combine MultinimialNB with GaussianNB to fit all the features

$$p(\mathbf{x} \mid C_k) = \frac{(\sum_i x_i)!}{\prod_i x_i!} \prod_i p_{ki}^{x_i} \quad \text{✖} \quad p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

3. Results:

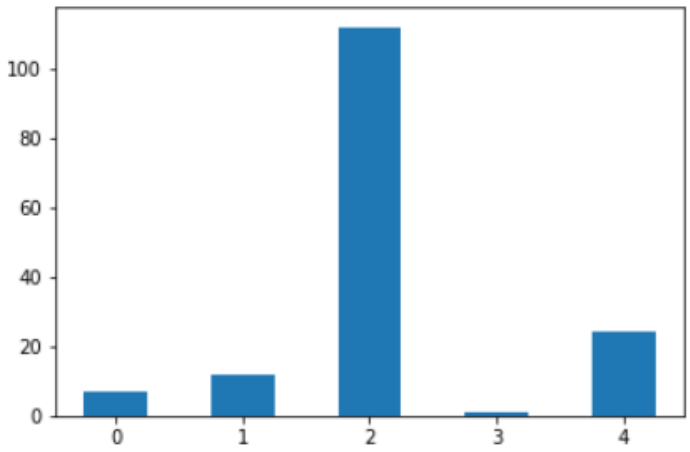
(1)Original Model:

```
clf.predict(x_test)
```

```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 2, 2, 2,
       2, 4, 4, 2, 4, 2, 2, 1, 4, 2, 2, 4, 1, 2, 2, 4, 4, 2, 4, 2, 0, 2, 4,
       1, 4, 3, 2, 2, 2, 2, 2, 2, 1, 2, 0, 0, 2, 4, 2, 1, 2, 2, 4, 2, 2, 0,
       2, 2, 2, 4, 1, 2, 2, 2, 2, 2, 4, 2, 2, 1, 2, 2, 2, 2, 4, 2, 2, 2, 2,
       2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 1, 2, 2, 4, 4, 2, 2, 2, 2, 4, 2, 0, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 1, 2, 2, 0, 2, 2, 2, 1, 2, 2, 0, 1, 2, 2, 4])
```

```
plt.hist(clf.predict(x_test),bins=[0,1,2,3,4,5],rwidth=0.5,align='left')
```

```
(array([   7.,   12.,  112.,    1.,   24.]),
 array([0, 1, 2, 3, 4, 5]),
 <a list of 5 Patch objects>)
```



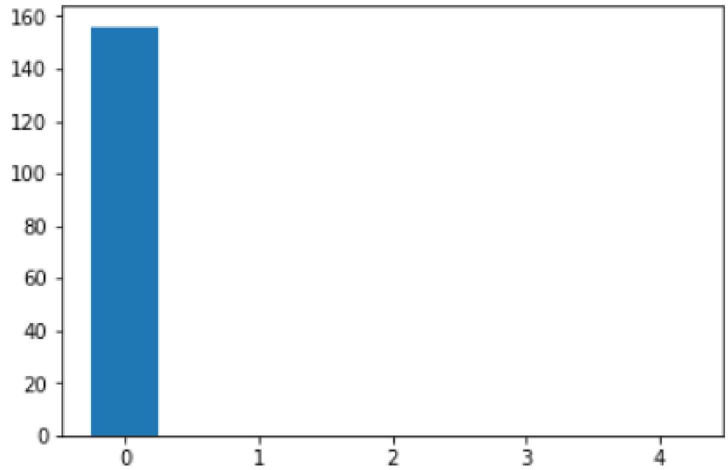
(1) MultinomialNB:

```
print clf_Laplace.predict_proba(x_cate_test).argmax(1)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0]
```

```
plt.hist(y_Laplace,bins=[0,1,2,3,4,5],rwidth=0.5,align='left')
```

```
(array([ 156.,    0.,    0.,    0.,    0.]),
 array([0, 1, 2, 3, 4, 5]),
 <a list of 5 Patch objects>)
```



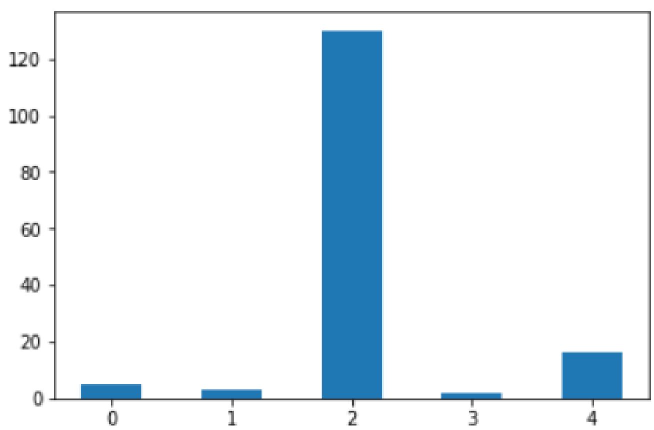
(2) GaussianNB:

```
clf.predict(x_test)
```

```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 4, 2, 2, 2,
       2, 4, 4, 2, 4, 2, 2, 1, 4, 2, 2, 4, 1, 2, 2, 4, 4, 2, 4, 2, 0, 2, 4,
       1, 4, 3, 2, 2, 2, 2, 2, 2, 1, 2, 0, 0, 2, 4, 2, 1, 2, 2, 4, 2, 2, 0,
       2, 2, 2, 4, 1, 2, 2, 2, 2, 2, 4, 2, 2, 1, 2, 2, 2, 2, 4, 2, 2, 2, 2,
       2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 1, 2, 2, 4, 4, 2, 2, 2, 2, 4, 2, 0, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 1, 2, 2, 0, 2, 2, 2, 1, 2, 2, 0, 1, 2, 2, 4])
```

```
plt.hist(y_Gaussian,bins=[0,1,2,3,4,5],rwidth=0.5,align='left')
```

```
(array([   5.,    3.,  130.,    2.,   16.]),
 array([0, 1, 2, 3, 4, 5]),
 <a list of 5 Patch objects>)
```



(3) Mixed Model:

```
y_pred = (clf_Gaussian.predict_proba(x_conti_test)*clf_Laplace.predict_proba(x_cate_test)/clf_Gaussian.class_prior_).argmax(1)
y_pred
```

```
array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 4, 4, 2, 4, 2, 2, 2, 4, 2, 2, 4, 2, 2, 1, 4, 2, 2, 2, 2, 2, 2, 4,
       2, 4, 3, 2, 2, 2, 2, 2, 2, 1, 2, 0, 0, 2, 2, 2, 2, 2, 2, 4, 2, 2, 3,
       2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 4, 2, 0, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 4, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 4, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 0, 1, 2, 2, 2])
```

```
print 'MultinomialNB model score:',accuracy_score(y_test_class, y_Laplace)
print 'GaussianNB model score:',accuracy_score(y_test_class, y_Gaussian)
print 'Mixed model score:',accuracy_score(y_test_class, y_pred)
```

```
MultinomialNB model score: 0.5
GaussianNB model score: 0.269230769231
Mixed model score: 0.269230769231
```

Comparison:

When we count the probability, the **prior** value will be included. In forestfires dataset, **most of the data are classified to be (area)class 0, owing to the concentrative distribution, 0 becomes the dominator.** Therefore, the skew data makes most of the predicions fall on class 0, jast as Grapgh y_Laplace shows).

But for the **GaussianNB classifier and our mixed classifier, the results become more normal**, we have predictions from class 0 to 4, which are closer to our real life distribution.