Fall 2018 MATH895 HW4

Qinghong(Jackie) Xu

December 4, 2018

1 Ex9.6

Th followings are the trees we fit for the ozone data set, as we can see, the second variable $X[1]$ and the third one $X[2]$ are more important, they are corresponding to the 'temperature' and 'wind' in our data set.

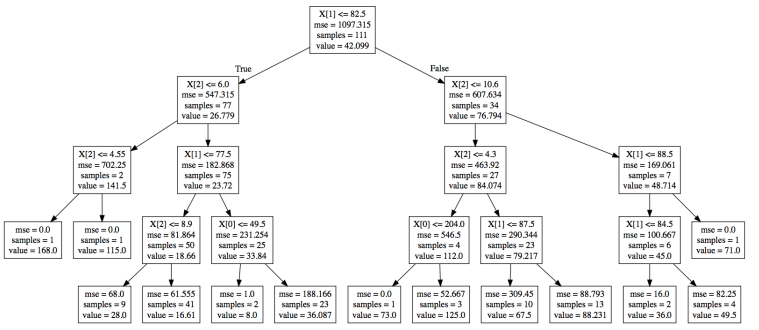


Figure 1: maximum depth = 4

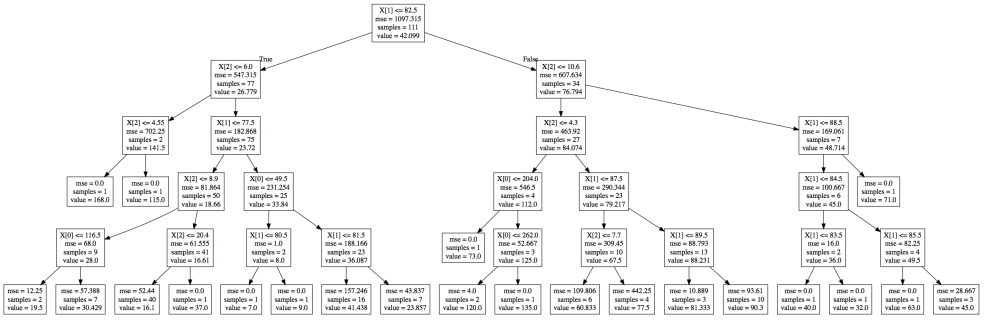


Figure 2: maximum depth = 5

2 Ex5.3

This plot compare the pointwise variance for different models. The variance near the boundary is large clearly, this is extremely obvious for the cubic splines. The natural cubic spline adds

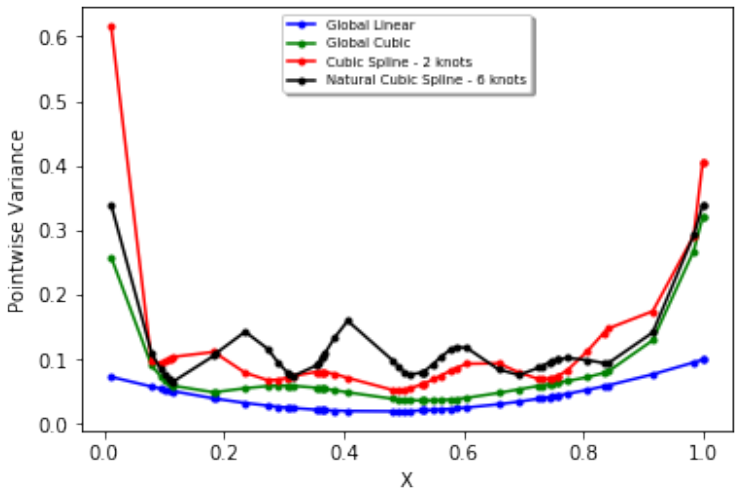


Figure 3: Pointwise variance curves for four different models, with X consisting of 50 points drawn at random from $U[0,1]$, and an assumed error model $\mathcal{N}(0,1)$. The linear and cubic polynomial fits have two and four degrees of freedom, respectively, while the cubic spline and natural cubic spline each have six degrees of freedom. The cubic spline has two knots at 0.33 and 0.66, while the natural spline has boundary knots at 0.1 and 0.9, and four interior knots uniformly spaced between them.

additional constraints, namely that the function is linear beyond the boundary knots. This frees up four degrees of freedom.Therefore, there are more knots in the interior region. We can see the trade off of variance in the plot. But there will be a price paid in bias near the boundary.

3 Ex7.6

Suppose we stack the outcomes $y_1, y_2, ..., y_N$ into a vector y, and similarly for the predictions $\hat{y}$ then a linear fitting model is

$$\hat{y} = Sy$$

where S is an $N \times N$ matrix. Then the effective number of parameters is defined as

$$df(S) = \text{trace}(S)$$

For k-NN method, $\hat{y}_i = \frac{1}{k}\sum_{x_i \in N_k(x)} y_i$. Since (x_i, y_i) will be in its neighbour, we know that the diagonal entries of S are all ones. Therefore

$$df(S) = \text{trace}(S) = \frac{N}{k}$$

4 Ex7.10

The cross-validation error is 50%. Suppose we have N observations and p features. There are two classes so the target will be either 0 or 1. If all of the observations are binary, then they are already classified so we don't need to estimate split points. If p is large enough, we could probably find a feature that has the same number as target, so we can split the entire training set perfectly. And we might hope we would split the validation data (one-fifth of data) perfectly as well.

Our experiment generates binary x randomly with 100 dimensions. This will be one of our feature. We also generates binary y randomly with 100 dimensions. This will be our target vector. We fit the data x with a stump and compute the misclassification error. We do 5-fold cross validation and compute the average of the error, the result is 50%. So the argument here is not correct. The experiment is attached in the end of the report.

MATH895hw4

December 2, 2018

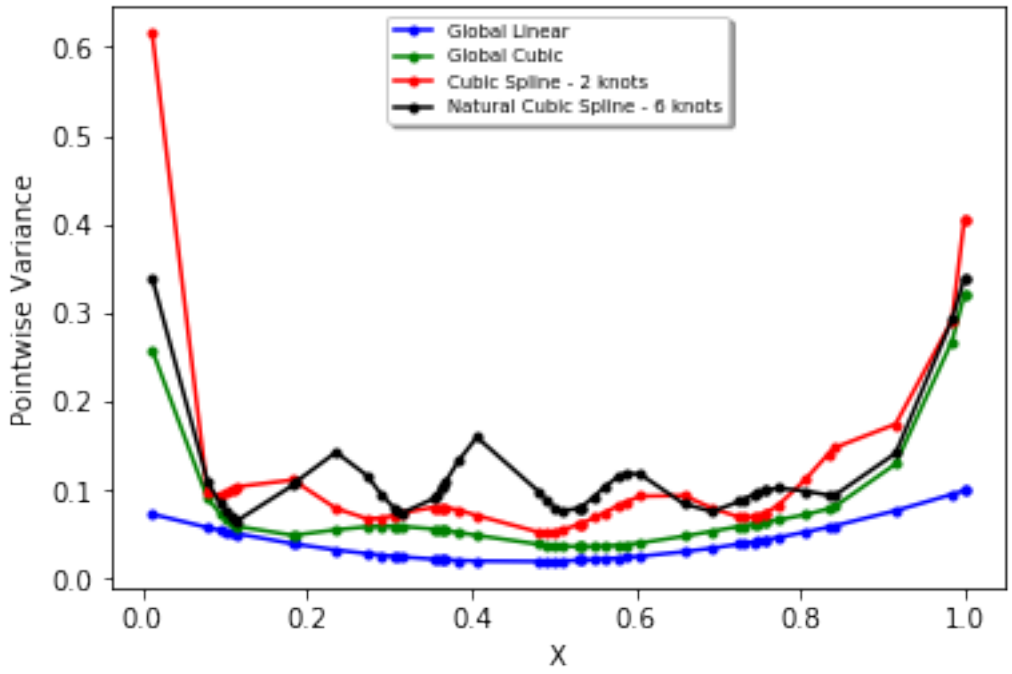
```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as pl

In [2]: from patsy import dmatrix
        import statsmodels.api as sm
        import statsmodels.formula.api as smf

In [19]: x = np.random.uniform(0, 1, 50)
         x = -np.sort(-x)
         predict_glinear = list()
         predict_gcubic = list()
         predict_cs2 = list()
         predict_cs6 = list()

In [20]: for i in range(100):
             y = np.random.normal(0, 1, 50)

             # Global linear
             coe_linear = np.polyfit(x, y, 1)
             linear = np.poly1d(coe_linear)
             predict_linear = linear(x)
             predict_glinear.append(predict_linear.tolist())

             # Global cubic
             coe_cubic = np.polyfit(x, y, 3)
             cubic = np.poly1d(coe_cubic)
             predict_cubic = cubic(x)
             predict_gcubic.append(predict_cubic.tolist())

             # Cubic spline 2 knots
             transformed_x = dmatrix("bs(x, knots=(0.33, 0.66), degree=3, include_intercept=Fa
             fit1 = sm.GLM(y, transformed_x).fit()
             predict_cs = fit1.predict(dmatrix("bs(x, knots=(0.33, 0.66), degree=3, include_in
             predict_cs2.append(predict_cs)

             #Natural cubic spline
             transformed_x = dmatrix("cr(x, df = 6, lower_bound = 0.1, upper_bound = 0.9, knot
```

```
        fit1 = sm.GLM(y, transformed_x).fit()
        predict_ncs = fit1.predict(dmatrix("cr(x, df = 6, lower_bound = 0.1, upper_bound =
        predict_cs6.append(predict_ncs)
```

In [21]:
```
var_glieanr = np.var(predict_glinear,0)
var_gcubic = np.var(predict_gcubic,0)
var_cs2 = np.var(predict_cs2,0)
var_cs6 = np.var(predict_cs6,0)
```

In [22]:
```
fig, ax = pl.subplots()
ax.plot(x,var_glieanr,'b.-', label = 'Global Linear')
ax.plot(x,var_gcubic,'g.-', label = 'Global Cubic')
ax.plot(x,var_cs2,'r.-', label = 'Cubic Spline - 2 knots')
ax.plot(x,var_cs6,'k.-', label = 'Natural Cubic Spline - 6 knots')
legend = ax.legend(loc='upper center', shadow=True, fontsize='x-small')
pl.xlabel('X')
pl.ylabel('Pointwise Variance')
```

Out[22]:
```
Text(0,0.5,'Pointwise Variance')
```



In [20]:
```
from sklearn import tree
from sklearn import cross_validation
from random import randrange
```

```
In [95]: x = np.random.choice([0, 1], size=(100,))
         y = np.random.choice([0, 1], size=(100,))
         dataset = np.column_stack((x,y))

In [96]: 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 [106]: def evaluate_cv(dataset, n_folds):
              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,[])

                  actual = [row[-1] for row in fold]
                  tree_model = tree.DecisionTreeClassifier(max_depth=1)

                  train_x = [[row[0]] for row in train_set]
                  train_y = [[row[-1]] for row in train_set]

                  model = tree_model.fit(train_x,train_y)
                  test_x = [[row[0]] for row in fold]
                  prediction = model.predict(test_x)
                  error = sum(abs(prediction - actual))/float(20)
              scores.append(error)
              return scores

In [108]: evaluate_cv(dataset.tolist(), 5)

Out[108]: [0.5]
```