rec11

November 12, 2020

1 CS 1656 – Introduction to Data Science

1.1 Instructor: Alexandros Labrinidis / Teaching Assistant: Evangelos Karageorgos

1.1.1 Additional credits: Xiaoting Li, Tahereh Arabghalizi, Evangelos Karageorgos, Zuha Agha, Anatoli Shein, Phuong Pham

1.2 Recitation 11: Regression and Decision Trees

In this recitation, we will learn how to do regression and classification with decision trees using scikit-learn python package.

```
In [1]: import pandas as pd
        import numpy as np
        from sklearn import linear_model, tree, metrics
        import matplotlib.pyplot as plt
        %matplotlib inline
```

1.3 Linear Regression

LinearRegression fits a linear model with coefficients w = (w_1, ..., w_p) to minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation.

LinearRegression will take in its fit method arrays X, y and will store the coefficients w of the linear model in its coef_ member.

We will now go through an example of linear regression on bike sharing dataset.

```
In [2]: df = pd.read_csv('http://data.cs1656.org/bike_share.csv')
        df.head()
```

Out[2]:

	instant	season	hr	holiday	weekday	workingday	weathersit	temp	temp_feels	hum	windspeed	cnt
0	1	1	0	0	6	0	1	0.24	0.2879	0.81	0.0	16
1	2	1	1	0	6	0	1	0.22	0.2727	0.80	0.0	40
2	3	1	2	0	6	0	1	0.22	0.2727	0.80	0.0	32
3	4	1	3	0	6	0	1	0.24	0.2879	0.75	0.0	13
4	5	1	4	0	6	0	1	0.24	0.2879	0.75	0.0	1

```
1      0.2727  0.80        0.0   40
2      0.2727  0.80        0.0   32
3      0.2879  0.75        0.0   13
4      0.2879  0.75        0.0    1
```

The attributes of the dataset are as follows:

```
- instant: record index
- season : season (1:spring, 2:summer, 3:fall, 4:winter)
- hr : hour (0 to 23)
- holiday : weather day is holiday or not (extracted from http://dchr.dc.gov/page/holiday-schedu
- weekday : day of the week (0 to 6)
- workingday : if day is neither weekend nor holiday is 1, otherwise is 0.
+ weathersit :
    - 1: Clear
    - 2: Misty, Cloudy
    - 3: Light Snow, Light Rain
    - 4: Heavy Rain, Ice Pallets
- temp : Normalized temperature in Celsius. The values are divided by 41 (max)
- temp_feels: Normalized feeling temperature in Celsius. The values are divided by 50 (max)
- hum: Normalized humidity. The values are divided by 100 (max)
- windspeed: Normalized wind speed. The values are divided by 67 (max)
- cnt: count of total rental bikes including both casual and registered
```

Our target variable, `y`, is *cnt*. We will use a single attribute as input feature for this example and will select *temp* as our input feature `X`. You will be using all attributes in one of your tasks.

1.3.1 Subsample

As our dataset consists of more than 17000 rows, we will randomly subsample our dataset to select 1000 rows.

```
In [3]: df_subsample = df.sample(1000)
        df_subsample.head()
```

```
Out[3]:        instant  season  hr  holiday  weekday  workingday  weathersit  temp  \
        6593      6594       4   7        0        5           1           1  0.42
        5093      5094       3   3        0        5           1           1  0.64
        3509      3510       2   3        0        2           1           2  0.64
        12988    12989       3   9        0        6           0           1  0.64
        16142    16143       4   8        0        6           0           1  0.32

               temp_feels   hum  windspeed  cnt
        6593       0.4242  0.94     0.0000  242
        5093       0.5909  0.78     0.0896    6
        3509       0.5606  0.94     0.0000    4
        12988      0.5758  0.89     0.1642  275
        16142      0.3485  0.76     0.0000  146
```

1.3.2 Train & Test Split

We will split the subsample into 90% training set and 10% test set by slicing the first 900 rows for training and using the rest for testing. As fit takes numpy arrays as input we will use *values* function to convert our Dataframe column into numpy array and use double brackets in order to make the arrays two-dimensional.

```
In [4]: train = df_subsample.iloc[1:900]
        train_x = train[['temp']].values
        train_y = train[['cnt']].values

        test=df_subsample.iloc[901:]
        test_x = test[['temp']].values
        test_y = test[['cnt']].values
        print (type(train_x), type(train_y), type(test_x), type(test_y))
```

```
<class 'numpy.ndarray'> <class 'numpy.ndarray'> <class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

1.3.3 Fit

To fit our linear regression model, apply the following function. Note that the fit function takes numpy array of the format [num_samples,num_features].

```
In [5]: # Create linear regression object
        regr = linear_model.LinearRegression()

        # Train the model using the training sets
        regr.fit(train_x, train_y)
```

```
Out[5]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

Now that we have fit our linear regression model onto the training data, our estimated model coefficients are stored in *coeff* attribute of our model.

```
In [6]: # The coefficients
        print('Coefficients: \n', regr.coef_)
```

```
Coefficients:
 [[363.82601247]]
```

1.3.4 Predict

We will now use our trained linear regression model to make predictions on our test set. Our model will take temperature attribute, *temp*, of our test data and will make predictions on the count of people who are bike sharing, given by *cnt*.

```
In [7]: predict_y = regr.predict(test_x)
        # Printing  predicted and actual values side by side fro comparison
        np.column_stack((predict_y,test_y))
```

```
Out[7]: array([[250.61406144, 179.        ],
               [ 83.2540957 ,  23.        ],
               [170.57233869, 331.        ],
               [185.12537919, 606.        ],
               [243.33754119, 185.        ],
               [316.10274368, 168.        ],
               [279.72014243, 441.        ],
               [279.72014243, 556.        ],
               [141.4662577 ,  46.        ],
               [243.33754119,  20.        ],
               [301.54970318, 419.        ],
               [323.37926393, 180.        ],
               [265.16710193, 257.        ],
               [286.99666268, 123.        ],
               [ 68.7010552 , 129.        ],
               [156.0192982 ,  42.        ],
               [192.40189944, 104.        ],
               [ 68.7010552 ,  47.        ],
               [243.33754119, 153.        ],
               [265.16710193,  14.        ],
               [301.54970318, 241.        ],
               [337.93230443, 129.        ],
               [ 97.8071362 ,  21.        ],
               [ 90.53061595, 275.        ],
               [272.44362218, 577.        ],
               [ 68.7010552 ,  37.        ],
               [ 90.53061595,   8.        ],
               [236.06102094, 306.        ],
               [170.57233869,  18.        ],
               [206.95493994, 456.        ],
               [236.06102094,  64.        ],
               [286.99666268, 306.        ],
               [236.06102094, 141.        ],
               [112.3601767 ,  63.        ],
               [228.78450069, 104.        ],
               [105.08365645,  69.        ],
               [243.33754119, 486.        ],
               [134.18973745,  16.        ],
               [301.54970318, 274.        ],
               [221.50798044, 245.        ],
               [294.27318293, 209.        ],
               [228.78450069, 204.        ],
               [177.84885894, 309.        ],
               [126.9132172 , 227.        ],
               [279.72014243, 131.        ],
               [286.99666268,  12.        ],
               [141.4662577 ,   4.        ],
               [148.74277795,   4.        ],
```

```
[279.72014243, 291.        ],
[221.50798044, 190.        ],
[163.29581844,  93.        ],
[ 83.2540957 , 679.        ],
[148.74277795,  53.        ],
[236.06102094,  21.        ],
[286.99666268,  53.        ],
[308.82622343, 504.        ],
[134.18973745,  40.        ],
[265.16710193, 299.        ],
[272.44362218, 226.        ],
[250.61406144, 243.        ],
[112.3601767 ,   4.        ],
[279.72014243, 292.        ],
[250.61406144,  19.        ],
[236.06102094,  86.        ],
[228.78450069,  14.        ],
[134.18973745, 296.        ],
[199.67841969, 634.        ],
[177.84885894, 184.        ],
[119.63669695, 136.        ],
[134.18973745, 137.        ],
[177.84885894, 233.        ],
[126.9132172 , 148.        ],
[148.74277795, 353.        ],
[265.16710193, 344.        ],
[170.57233869, 157.        ],
[ 90.53061595,  20.        ],
[177.84885894, 114.        ],
[112.3601767 ,  27.        ],
[265.16710193,  11.        ],
[ 83.2540957 , 110.        ],
[148.74277795, 228.        ],
[ 61.42453495,  57.        ],
[177.84885894, 292.        ],
[243.33754119, 186.        ],
[185.12537919,  21.        ],
[192.40189944, 196.        ],
[170.57233869,  18.        ],
[236.06102094,  45.        ],
[272.44362218, 237.        ],
[199.67841969, 374.        ],
[105.08365645, 178.        ],
[177.84885894, 398.        ],
[199.67841969,  12.        ],
[243.33754119, 308.        ],
[156.0192982 ,  27.        ],
[185.12537919,  21.        ],
```

```
[214.23146019,  10.        ],
[177.84885894,  32.        ],
[ 97.8071362 ,  57.        ]])
```

1.3.5 Mean Squared Error

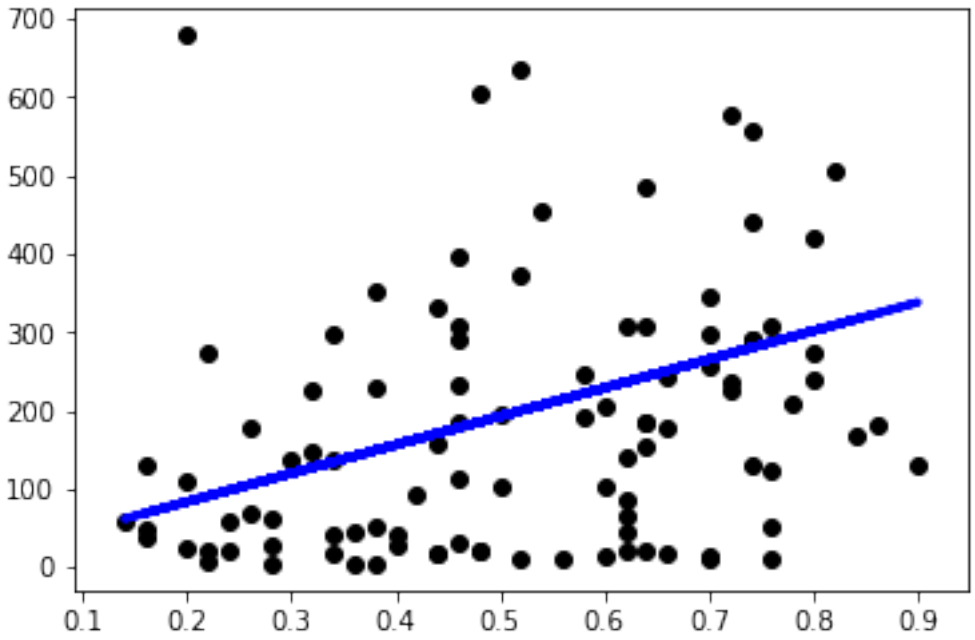
Looks like some of our model's predictions are not good. Now, let's measure the difference between our predicted and actual values by calculating the mean squared error.

```
In [8]: meansq_error = np.mean((predict_y - test_y) ** 2)
        print ("Mean squared error: %.2f" % meansq_error)
```

```
Mean squared error: 25105.38
```

As expected our mean squared error is high, which means our model is not good. Can we improve it? What if we use more training data? Or more features? ### Plot We can also visualize the difference between our predictions and actual values by plotting.

```
In [9]: plt.scatter(test_x, test_y,  color='black', linewidth=1)
        plt.plot(test_x, predict_y, color='blue', linewidth=3)
        plt.show()
```



1.4 Decision Trees

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

We will go through an example of binary classfication using decision trees on titanic survival dataset.

```
In [10]: dt = pd.read_csv('http://data.cs1656.org/titanic.csv')
         dt.head()
```

```
Out[10]:    PassengerId  Survived  Pclass  \
         0            1         0       3
         1            2         1       1
         2            3         1       3
         3            4         1       1
         4            5         0       3

                                                      Name     Sex   Age  SibSp  \
         0                            Braund, Mr. Owen Harris    male  22.0      1
         1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
         2                             Heikkinen, Miss. Laina  female  26.0      0
         3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
         4                           Allen, Mr. William Henry    male  35.0      0

            Parch Embarked
         0      0        S
         1      0        C
         2      0        S
         3      0        S
         4      0        S
```

The attributes of the dataset are as follows: - survival Survival (0 = No; 1 = Yes) - pclass Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd) - name Name - sex Sex - age Age - sibsp Number of Siblings/Spouses Aboard - parch Number of Parents/Children Aboard - embarked Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton) Our target class variable is *Survived*, whether the passenger survived or not. We will use only a subset of attributes that take discreet values to build our decision tree.

To fit a decision tree model, we will have to convert the categoricalvalues into numerical values. As the only categorical attribute we will use is *Sex*, we will only need to convert that column into numerical values using the following commands.

```
In [11]: dt['Sex'] = dt['Sex'].replace(['female', 'male'], [1, 2])
         dt.head()
```

```
Out[11]:    PassengerId  Survived  Pclass  \
         0            1         0       3
         1            2         1       1
         2            3         1       3
         3            4         1       1
         4            5         0       3

                                                      Name  Sex   Age  SibSp  Parch  \
         0                            Braund, Mr. Owen Harris    2  22.0      1      0
```

```
1  Cumings, Mrs. John Bradley (Florence Briggs Th...    1  38.0      1      0
2                             Heikkinen, Miss. Laina    1  26.0      0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)    1  35.0      1      0
4                           Allen, Mr. William Henry    2  35.0      0      0

  Embarked
0        S
1        C
2        S
3        S
4        S
```

1.4.1 Train & Test Split

We will split our data into train and test set using the first 800 rows for training and the rest for testing.

```
In [12]: dt_train_x = dt.iloc[:800][['Pclass','Sex','SibSp']].values
         dt_train_y = dt.iloc[:800][['Survived']].values

         dt_test_x = dt.iloc[801:][['Pclass','Sex','SibSp']].values
         dt_test_y = dt.iloc[801:][['Survived']].values
```

1.4.2 Fit

We will now fit our decision tree model onto the training set.

```
In [13]: clf = tree.DecisionTreeClassifier()
         clf = clf.fit(dt_train_x, dt_train_y)
```

1.4.3 Predict

```
In [14]: dt_predict_y = clf.predict(dt_test_x)
         ## comparing predicted and actual values
         np.column_stack((dt_predict_y,dt_test_y))
```

```
Out[14]: array([[1, 1],
                [0, 1],
                [0, 1],
                [0, 1],
                [0, 0],
                [0, 0],
                [1, 0],
                [0, 0],
                [1, 1],
                [0, 0],
                [0, 0],
                [0, 0],
                [0, 0],
```

```
[0, 0],
[0, 0],
[1, 0],
[0, 0],
[0, 0],
[0, 0],
[1, 1],
[0, 1],
[0, 0],
[1, 1],
[0, 0],
[0, 0],
[0, 0],
[0, 1],
[0, 1],
[1, 1],
[0, 1],
[0, 1],
[0, 0],
[0, 0],
[0, 0],
[1, 1],
[0, 0],
[0, 0],
[0, 1],
[0, 1],
[0, 0],
[0, 0],
[1, 1],
[0, 0],
[0, 0],
[0, 0],
[0, 0],
[0, 0],
[0, 0],
[1, 1],
[0, 0],
[0, 0],
[0, 0],
[1, 1],
[1, 0],
[1, 1],
[1, 1],
[0, 1],
[1, 1],
[0, 0],
[0, 0],
[0, 0],
```

```
       [1, 1],
       [0, 0],
       [0, 0],
       [1, 1],
       [1, 1],
       [0, 0],
       [0, 0],
       [0, 1],
       [0, 0],
       [1, 1],
       [0, 0],
       [0, 0],
       [1, 1],
       [1, 1],
       [0, 0],
       [0, 0],
       [0, 0],
       [1, 1],
       [1, 1],
       [0, 0],
       [1, 0],
       [0, 0],
       [0, 0],
       [1, 0],
       [0, 0],
       [1, 1],
       [0, 0],
       [0, 1],
       [0, 0]], dtype=int64)
```

1.4.4 Accuracy

We can measure the accuracy of our prediction by using the following commands.

```
In [15]: accuracy = metrics.accuracy_score(dt_test_y,dt_predict_y)
         accuracy
```

```
Out[15]: 0.8
```

1.5 Tasks

** Task 1**

Do linear regression over a sample of 1000 rows of bike share counts, *cnt*, using *weekday*, as input feature. Calculate the mean squared error by using first 900 rows for training and the rest for testing. Return the mean squared error.

** Task 2.1**

Repeat Task 1 using all atttributes except instant (also, scatter plot is not required in this task). Is the mean squared error higher or lower? Is it better to use all attributes?

** Task 2.2**

Comparing the results of task 1 and task 2.1, is it better to use all attributes? Why?

** Task 3**

You will use bank-data.csv as input for this task. Use decision trees to do binary classification of mortgage{yes,no} using region, sex and married attributes as input features. Use the first 500 rows for training and the rest for testing. Measure the accuracy of your classification. Return the accuracy.