EDA_video3_screencast

December 23, 2019

IMPORTANT: You will not be able to run this notebook at coursera platform, as the dataset is not there. The notebook is in read-only mode.

But you can run the notebook locally and download the dataset using this link to explore the data interactively.

```
In [2]: pd.set_option('max_columns', 100)
```

1 Load the data

```
In [3]: train = pd.read_csv('./train.csv')
        train.head()
```

Out[3]:

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x21	x22	x23	x24	x25
0	b4d8a653ea	16a14a2d17	06330986ed	ca63304de0	a62168d626	1746600cb0	1	1	-0.688706	7e5c97705a	e5df3eff9b	91bb549494	e33c63cf35	3694.0	6e40247e69	617a4ad3f9	718c61545b	c26d08129a	634e3cf3ac	dd9c9e0da2	17c99905b6	513a3e3f36	9aba4d7f51	40.579612	-0.112693	-0.172191
1	467f9617a3	16a14a2d17	06330986ed	ca63304de0	b7584c2d52	1746600cb0	1	1	0.870871	5624b8f759	fa0b797a92	669ea3d319	f178803074	18156.0	01ede04b4b	617a4ad3f9	718c61545b	d342e2765f	bb20e1ca06	8a6c8cef83	1b02793146	992153ed65	9aba4d7f51	28.765503	2.612285	2.159091
2	190436e528	16a14a2d17	06330986ed	ca63304de0	b7584c2d52	1746600cb0	1	1	0.437655	5624b8f759	152af2cb2f	91bb549494	e33c63cf35	1178.0	cc69cbe29a	617a4ad3f9	e8a040423a	c82c3dbd33	ee3501282b	199ce7c484	5f17dedd5c	5c5025bd0a	9aba4d7f51	24.943933	-0.814660	-0.708308
3	43859085bc	16a14a2d17	06330986ed	ca63304de0	a62168d626	1746600cb0	1	1	0.004439	f67f142e40	c4dd2197c3	91bb549494	e33c63cf35	14559.0	6e40247e69	617a4ad3f9	718c61545b	c26d08129a	9e166b965d	466f8951b0						
4	a4c3095b75	16a14a2d17	06330986ed	ca63304de0	b7584c2d52	1746600cb0	1	1	0.480977	7e5c97705a	e071d01df5	91bb549494	e33c63cf35	5777.0	6e40247e69	617a4ad3f9	4b9480aa42	e84655292c	527b6ca8cc	dd9c9e0da2						

```
3  fde72a6d5c  acfadc5c01  9aba4d7f51  41.576860 -0.907833 -0.761736
4  17c99905b6  0fc56ea1f0  9aba4d7f51  31.080282 -0.371787 -0.367616

        x26       x27       x28        x29          x30    x31       x32  \
0  1.166667  1.674538  0.630889  37.000000     1.294922   55.0  0.166667
1  4.000000  1.710714  1.713538   0.166667     0.027669  109.0  0.000000
2  1.500000 -0.512422 -0.733967   0.333333    14.837728   11.0  0.000000
3  0.500000 -0.627525 -0.805801   1.166667     0.004395    0.0  0.500000
4  1.666667  0.271307  0.013112  17.333333  1713.439128   33.0  0.000000

    x33  x34       x35  x36    x37  x38  x39   x40    x41   x42    x43  x44  \
0  10.0  0.0  0.000000  1.0    9.0  0.0  1.0  23.0   3.67  0.12  1.935  2.2
1  31.0  0.0  0.000000  1.0  244.0  1.0  1.0  68.0  17.25  0.57  3.452  4.0
2  24.0  0.0  0.000000  1.0   29.0  0.0  3.0  11.0   4.42  0.15  0.161  0.2
3   0.0  0.0  0.000000  7.0    7.0  0.0  3.0  15.0   8.92  0.29  0.226  0.8
4   6.0  1.0  0.666667  8.0  108.0  1.0  4.0  86.0   1.58  0.05  2.032  2.4

     x45    x46    x47    x48    x49    x50    x51    x52       x53       x54  \
0  0.625  0.250  0.125  0.000  0.813  0.074  0.634  0.548  0.235333  0.264952
1  0.409  0.619  0.579  0.248  0.346  0.541  0.522  0.000  1.782346  1.322409
2  1.000  1.000  1.000  1.000  1.000  0.520  0.533  0.835 -0.586540  0.672436
3  0.000  0.000  0.000  0.000  0.000  1.000  0.000  0.000 -1.600326 -1.838680
4  0.348  0.762  0.550  0.392  0.489  0.517  1.000  0.642  0.960991  0.790990

        x55       x56       x57       x58       x59       x60    x61  y
0  0.000000  0.333333  0.333333  0.333333  0.000000  0.000000    9.0  2
1  0.011647  0.397671  0.239601  0.249584  0.068220  0.033278  601.0  4
2  0.000000  0.606061  0.121212  0.212121  0.060606  0.000000   33.0  3
3  0.000000  1.000000  0.000000  0.000000  0.000000  0.000000    1.0  4
4  0.020161  0.645161  0.258065  0.036290  0.040323  0.000000  248.0  3
```

2 Build a quick baseline

In [4]:
```
from sklearn.ensemble import RandomForestClassifier

# Create a copy to work with
X = train.copy()

# Save and drop labels
y = train.y
X = X.drop('y', axis=1)

# fill NANs
X = X.fillna(-999)

# Label encoder
for c in train.columns[train.dtypes == 'object']:
```

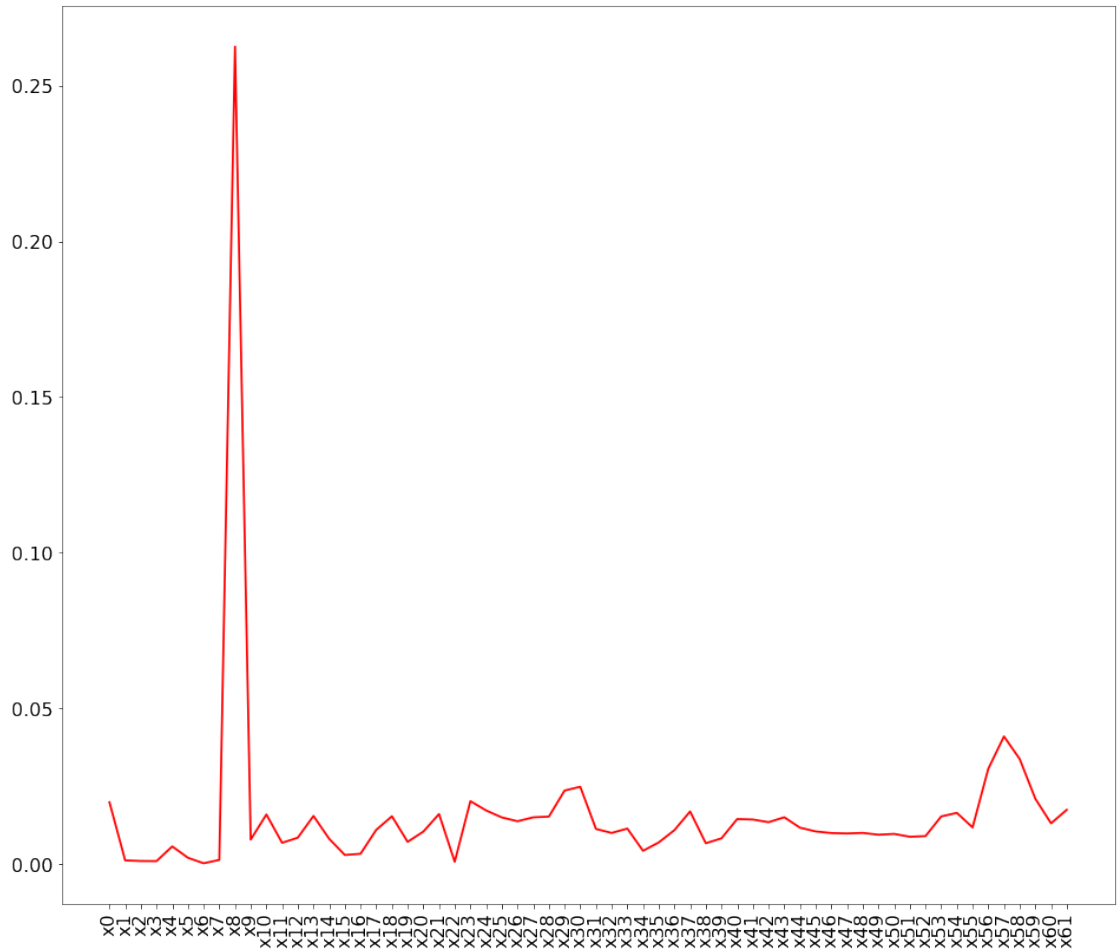
```
            X[c] = X[c].factorize()[0]

        rf = RandomForestClassifier()
        rf.fit(X,y)
```

```
Out[4]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                    max_depth=None, max_features='auto', max_leaf_nodes=None,
                    min_impurity_split=1e-07, min_samples_leaf=1,
                    min_samples_split=2, min_weight_fraction_leaf=0.0,
                    n_estimators=10, n_jobs=1, oob_score=False, random_state=None,
                    verbose=0, warm_start=False)
```

```
In [5]: plt.plot(rf.feature_importances_)
        plt.xticks(np.arange(X.shape[1]), X.columns.tolist(), rotation=90);
```

```
/home/dulyanov/miniconda2/lib/python2.7/site-packages/matplotlib/font_manager.py:1297: UserWarni
  (prop.get_family(), self.defaultFamily[fontext]))
```



There is something interesting about x8.

In [6]:
```
# we see it was standard scaled, most likely, if we concat train and test, we will get e
print 'Mean:', train.x8.mean()
print 'std:', train.x8.std()
```

```
Mean: -0.000252352028622
std: 1.02328163601
```

In [7]:
```
# And we see that it has a lot of repeated values
train.x8.value_counts().head(15)
```

Out[7]:
```
-2.984750    2770
 0.480977    2569
 0.610941    1828
 0.654263    1759
 0.567620    1746
 0.697585    1691
 0.524298    1639
 0.740906    1628
 0.394333    1610
 0.437655    1513
 0.351012    1450
 0.264369    1429
 0.307690    1401
 0.221047    1372
 0.784228    1293
Name: x8, dtype: int64
```

In [8]:
```
# It's very hard to work with scaled feature, so let's try to scale them back
# Let's first take a look at difference between neighbouring values in x8

x8_unique = train.x8.unique()
x8_unique_sorted = np.sort(x8_unique)

np.diff(x8_unique_sorted)
```

Out[8]:
```
array([ 43.27826527,  38.98942817,   0.21660793,   0.04332159,
         0.17328635,   0.21660793,   0.08664317,   0.04332159,
         0.04332159,   0.04332159,   0.04332159,   0.04332159,
         0.04332159,   0.04332159,   0.04332159,   0.04332159,
         0.04332159,   0.04332159,   0.04332159,   0.04332159,
         0.04332159,   0.04332159,   0.04332159,   0.04332159,
         0.04332159,   0.04332159,   0.04332159,   0.04332159,
         0.04332159,   0.04332159,   0.04332159,   0.04332159,
         0.04332159,   0.04332159,   0.04332159,   0.04332159,
         0.04332159,   0.04332159,   0.04332159,   0.04332159,
         0.04332159,   0.04332159,   0.04332159,   0.04332159,
```

```
       0.04332159,   0.04332159,   0.04332159,   0.04332159,
       0.04332159,   0.04332159,   0.04332159,   0.04332159,
       0.04332159,   0.04332159,   0.04332159,   0.04332159,
       0.04332159,   0.04332159,   0.04332159,   0.04332159,
       0.04332159,   0.04332159,   0.04332159,   0.04332159,
       0.04332159,   0.04332159,   0.04332159,   0.04332159,
       0.04332159,   0.04332159,   0.04332159,   0.04332159,
       0.04332159,   0.04332159,   0.04332159,   0.04332159,
       0.04332159,   0.04332159,   0.12996476,   0.04332159,
       0.04332159,   0.04332159,   0.04332159,   0.04332159,
       0.04332159,   0.04332159,   0.21660793,   1.16968285,
       0.04332159,   0.38989428,          nan])
```

In [9]:
```
# The most of the diffs are 0.04332159!
# The data is scaled, so we don't know what was the diff value for the original feature
# But let's assume it was 1.0
# Let's devide all the numbers by 0.04332159 to get the right scaling
# note, that feature will still have zero mean

np.diff(x8_unique_sorted/0.04332159)
```

Out[9]:
```
array([ 998.99992752,  899.9999347 ,    4.99999964,    0.99999993,
          3.99999971,    4.99999964,    1.99999985,    0.99999993,
          0.99999993,    0.99999993,    0.99999993,    0.99999993,
          0.99999993,    0.99999993,    0.99999993,    0.99999993,
          0.99999993,    0.99999993,    0.99999993,    0.99999993,
          0.99999993,    0.99999993,    0.99999993,    0.99999993,
          0.99999993,    0.99999993,    0.99999993,    0.99999993,
          0.99999993,    0.99999993,    0.99999993,    0.99999993,
          0.99999993,    0.99999993,    0.99999993,    0.99999993,
          0.99999993,    0.99999993,    0.99999993,    0.99999993,
          0.99999993,    0.99999993,    0.99999993,    0.99999993,
          0.99999993,    0.99999993,    0.99999993,    0.99999993,
          0.99999993,    0.99999993,    0.99999993,    0.99999993,
          0.99999993,    0.99999993,    0.99999993,    0.99999993,
          0.99999993,    0.99999993,    0.99999993,    0.99999993,
          0.99999993,    0.99999993,    0.99999993,    0.99999993,
          0.99999993,    0.99999993,    2.99999978,    0.99999993,
          0.99999993,    0.99999993,    0.99999993,    0.99999993,
          0.99999993,    0.99999993,    4.99999964,   26.99999804,
          0.99999993,    8.99999935,          nan])
```

In [10]:
```
(train.x8/0.04332159).head(10)
```

```
Out[10]: 0   -15.897530
         1    20.102468
         2    10.102468
         3     0.102469
         4    11.102468
         5   -68.897526
         6    10.102468
         7    15.102468
         8     9.102468
         9   -68.897526
         Name: x8, dtype: float64

In [11]: # Ok, now we see .102468 in every value
         # this looks like a part of a mean that was subtracted during standard scaling
         # If we subtract it, the values become almost integers
         (train.x8/0.04332159 - .102468).head(10)

Out[11]: 0   -15.999998
         1    20.000000
         2    10.000000
         3     0.000001
         4    11.000000
         5   -68.999994
         6    10.000000
         7    15.000000
         8     9.000000
         9   -68.999994
         Name: x8, dtype: float64

In [12]: # let's round them
         x8_int = (train.x8/0.04332159 - .102468).round()
         x8_int.head(10)

Out[12]: 0   -16.0
         1    20.0
         2    10.0
         3     0.0
         4    11.0
         5   -69.0
         6    10.0
         7    15.0
         8     9.0
         9   -69.0
         Name: x8, dtype: float64

In [13]: # Ok, what's next? In fact it is not obvious how to find shift parameter,
         # and how to understand what the data this feature actually store
         # But ...
```

```
In [14]: x8_int.value_counts()
```

```
Out[14]: -69.0      2770
          11.0      2569
          14.0      1828
          15.0      1759
          13.0      1746
          16.0      1691
          12.0      1639
          17.0      1628
          9.0       1610
          10.0      1513
          8.0       1450
          6.0       1429
          7.0       1401
          5.0       1372
          18.0      1293
          1.0       1290
          4.0       1276
          2.0       1250
          3.0       1213
         -1.0       1085
          0.0       1080
         -2.0       1006
         -4.0        995
         -3.0        976
         -5.0        954
         -8.0        923
         -9.0        921
         -6.0        906
          19.0       893
         -7.0        881
                     ...
          26.0         3
         -40.0         3
         -41.0         3
          25.0         2
         -59.0         2
          31.0         2
          34.0         2
         -46.0         2
         -49.0         2
          33.0         2
         -42.0         2
          32.0         2
          37.0         2
          30.0         2
         -45.0         2
```

```
-54.0         1
 36.0         1
-51.0         1
 27.0         1
 79.0         1
-47.0         1
 69.0         1
 70.0         1
-50.0         1
-1968.0       1
 42.0         1
-63.0         1
-48.0         1
-64.0         1
 35.0         1
Name: x8, Length: 99, dtype: int64
```

In [ ]:
```
# do you see this -1968? Doesn't it look like a year? ... So my hypothesis is that this
# Maybe it was a textbox where users enter their year of birth, and someone entered 0000
# The hypothesis looks plausible, isn't it?
```

In [22]:
```
(x8_int + 1968.0).value_counts().sort_index()
```

Out[22]:
```
0.0          1
999.0        4
1899.0    2770
1904.0       1
1905.0       1
1909.0       2
1914.0       1
1916.0       3
1917.0       1
1918.0       1
1919.0       2
1920.0       1
1921.0       1
1922.0       2
1923.0       2
1924.0       4
1925.0       4
1926.0       2
1927.0       3
1928.0       3
1929.0       4
1930.0       4
1931.0      12
1932.0      10
1933.0       7
```

```
1934.0      13
1935.0      28
1936.0      35
1937.0      35
1938.0      45
           ...
1978.0    1513
1979.0    2569
1980.0    1639
1981.0    1746
1982.0    1828
1983.0    1759
1984.0    1691
1985.0    1628
1986.0    1293
1987.0     893
1988.0     624
1989.0     434
1990.0     233
1991.0     110
1992.0      31
1993.0       2
1994.0       3
1995.0       1
1998.0       2
1999.0       2
2000.0       2
2001.0       2
2002.0       2
2003.0       1
2004.0       1
2005.0       2
2010.0       1
2037.0       1
2038.0       1
2047.0       1
Name: x8, Length: 99, dtype: int64
```

In [23]: *# After the competition ended the organisers told it was really a year of birth*