step01-prudential-kaggle-151202

January 31, 2016

https://www.kaggle.com/c/prudential-life-insurance-assessment/data

In [1]:
```
import pandas as pd
pd.options.display.max_columns = None
%matplotlib inline
import numpy as np
```

0.1 Concatenate train and test data into one dataframe for feature extraction

In [2]:
```
train = pd.read_csv('input/train.csv')
test = pd.read_csv('input/test.csv')
sample_out = pd.read_csv('input/sample_submission.csv')

train['train?'] = True
test['train?'] = False

print("length of training set = {:d} observations".format(len(train)))
print("length of testing set = {:d} observations".format(len(test)))

data = pd.concat((train,test)).reset_index(drop=True)

assert len(data) == len(train) + len(test)
print("correctly concatenated train and test data  =>  num. observations = {:d}".format(len(data

data_features = data.drop(['Response', 'train?'], axis=1)
```

```
length of training set = 59381 observations
length of testing set = 19765 observations
correctly concatenated train and test data  =>  num. observations = 79146
```

0.2 List features that have null values:

In [3]:
```
features_with_nulls = {k:v for k, v in data_features.isnull().apply(sum).to_dict().items() if v
features_with_nulls
```

Out[3]:
```
{'Employment_Info_1': 22,
 'Employment_Info_4': 8916,
 'Employment_Info_6': 14641,
 'Family_Hist_2': 38536,
 'Family_Hist_3': 45305,
 'Family_Hist_4': 25861,
 'Family_Hist_5': 55435,
 'Insurance_History_5': 33501,
 'Medical_History_1': 11861,
```

```
'Medical_History_10': 78388,
'Medical_History_15': 59460,
'Medical_History_24': 74165,
'Medical_History_32': 77688}
```

Turns out that `Product_Info_2` in fact has categorical values. Is it the only feature with non-numeric values?

0.3 List features with non-numerical values

```
In [4]: # Is Product_Info_2 the only non-numerical feature?
        nonnum = {col: train.dtypes[col] for col in data_features.columns if data_features.dtypes[col] n
        print("non-mumerical columns = {}".format(nonnum))
```

```
non-mumerical columns = {'Product_Info_2': dtype('O')}
```

0.4 List features with values < 0 :

```
In [5]: # Features that have < 0 values:
        {k:v for k, v in (data_features < 0).apply(sum).to_dict().items() if v > 0}
```

```
Out[5]: {'Product_Info_2': 79146}
```

0.5 Take a closer look at the values in `Product_Info_2`:

```
In [6]: col = 'Product_Info_2'
        print("unique vals = {}".format(train[col].unique()))
        print("num unique = {}".format(len(train[col].unique())))
        print()

        for val in sorted(train[col].unique()):
            val_set = train[train[col]==val]
            print("\n{} - {} observations".format(val, len(val_set)))
            print(val_set.groupby('Response').size().to_dict())
```

```
unique vals = ['D3' 'A1' 'E1' 'D4' 'D2' 'A8' 'A2' 'D1' 'A7' 'A6' 'A3' 'A5' 'C4' 'C1' 'B2'
 'C3' 'C2' 'A4' 'B1']
num unique = 19


A1 - 2363 observations
{1: 132, 2: 235, 3: 51, 4: 68, 5: 195, 6: 352, 7: 270, 8: 1060}

A2 - 1974 observations
{1: 238, 2: 267, 3: 20, 4: 39, 5: 149, 6: 415, 7: 276, 8: 570}

A3 - 977 observations
{1: 70, 2: 104, 3: 17, 4: 31, 5: 64, 6: 156, 7: 124, 8: 411}

A4 - 210 observations
{1: 19, 3: 4, 4: 10, 5: 17, 6: 34, 7: 30, 8: 96}

A5 - 775 observations
{1: 53, 3: 13, 4: 28, 5: 42, 6: 126, 7: 138, 8: 375}
```

```
A6 - 2098 observations
{1: 122, 3: 26, 4: 44, 5: 122, 6: 327, 7: 296, 8: 1161}

A7 - 1383 observations
{1: 281, 2: 285, 3: 94, 4: 4, 5: 662, 6: 12, 7: 15, 8: 30}

A8 - 6835 observations
{1: 953, 2: 728, 3: 110, 4: 89, 5: 927, 6: 1086, 7: 965, 8: 1977}

B1 - 54 observations
{1: 7, 3: 9, 4: 6, 5: 8, 6: 11, 7: 5, 8: 8}

B2 - 1122 observations
{1: 74, 2: 38, 3: 11, 4: 21, 5: 76, 6: 182, 7: 173, 8: 547}

C1 - 285 observations
{1: 45, 2: 40, 3: 9, 4: 13, 5: 19, 6: 47, 7: 39, 8: 73}

C2 - 160 observations
{1: 22, 2: 28, 3: 5, 4: 5, 5: 11, 6: 25, 7: 24, 8: 40}

C3 - 306 observations
{1: 33, 2: 28, 3: 10, 4: 10, 5: 24, 6: 61, 7: 48, 8: 92}

C4 - 219 observations
{1: 19, 2: 17, 3: 2, 4: 7, 5: 14, 6: 35, 7: 27, 8: 98}

D1 - 6554 observations
{1: 1065, 2: 1275, 3: 169, 4: 277, 5: 435, 6: 1316, 7: 738, 8: 1279}

D2 - 6286 observations
{1: 746, 2: 917, 3: 110, 4: 154, 5: 423, 6: 1542, 7: 921, 8: 1473}

D3 - 14321 observations
{1: 1440, 2: 1675, 3: 237, 4: 420, 5: 1256, 6: 3281, 7: 2080, 8: 3932}

D4 - 10812 observations
{1: 687, 2: 707, 3: 82, 4: 124, 5: 810, 6: 1797, 7: 1457, 8: 5148}

E1 - 2647 observations
{1: 201, 2: 208, 3: 34, 4: 78, 5: 178, 6: 428, 7: 401, 8: 1119}
```

It seems that the values of `Product_Info_2` have no ordering, so at the end we will keep this in mind and make it a one-hot encoded variable.

0.6 Now, let's take a closer look at the features that have null values

To do this we start by defining a function that allows us to take a quick glance at a particular feature

In [7]:
```
import io

def explore_feature(dataframe, col):
    with io.StringIO("") as fout:
```

```
            print("\n\n" + "="*50, file=fout)
            print("column name = {}, dtype = {}".format(col, dataframe.dtypes[col]), file=fout)
            print("-"*40, file=fout)
            print("value counts:", file=fout)
            if len(dataframe[col].unique()) > 50:
                print("\t>> more than 50 unique values <<", file=fout)
            else:
                low_frequency_values = []
                for index,value in dataframe[col].value_counts().iteritems():
                    if value > 1000:
                        print("{:s} {:s}".format(str(index).ljust(10), str(value).rjust(6)), file=f
                    else:
                        low_frequency_values.append("{0}({1})".format(str(index), value))
                if len(low_frequency_values) > 0:
                    print("values with < 1000 counts: [{}]".format(', '.join(low_frequency_values))

            print("-"*50, file=fout)
            stats = dataframe[col].describe()
            try:
                strstats = ', '.join("{}:{:.2g}".format(k, float(v)) for k, v in stats.items() if k
            except:
                strstats = None


            if strstats is not None:
                print(strstats, file=fout)

            column_info = fout.getvalue()
            return column_info
```

Using our `explore_feature` function let's take a close look at those features that have null values:

```
In [8]: for k, v in features_with_nulls.items():
            print(explore_feature(data, k))
```

```
==================================================
column name = Employment_Info_4, dtype = float64
----------------------------------------
value counts:
        >> more than 50 unique values <<
--------------------------------------------------
mean:0.0063, std:0.033, min:0, 25%:0, 50%:0, 75%:0, max:1



==================================================
column name = Medical_History_24, dtype = float64
----------------------------------------
value counts:
        >> more than 50 unique values <<
--------------------------------------------------
mean:50, std:78, min:0, 25%:1, 50%:8, 75%:63, max:2.4e+02
```

```
====================================================
column name = Family_Hist_4, dtype = float64
----------------------------------------
value counts:
        >> more than 50 unique values <<
----------------------------------------------------
mean:0.45, std:0.16, min:0, 25%:0.32, 50%:0.44, 75%:0.56, max:1



====================================================
column name = Employment_Info_1, dtype = float64
----------------------------------------
value counts:
        >> more than 50 unique values <<
----------------------------------------------------
mean:0.078, std:0.083, min:0, 25%:0.035, 50%:0.06, 75%:0.1, max:1



====================================================
column name = Medical_History_15, dtype = float64
----------------------------------------
value counts:
        >> more than 50 unique values <<
----------------------------------------------------
mean:1.2e+02, std:99, min:0, 25%:18, 50%:1.2e+02, 75%:2.4e+02, max:2.4e+02



====================================================
column name = Employment_Info_6, dtype = float64
----------------------------------------
value counts:
        >> more than 50 unique values <<
----------------------------------------------------
mean:0.36, std:0.35, min:0, 25%:0.06, 50%:0.25, 75%:0.58, max:1



====================================================
column name = Medical_History_10, dtype = float64
----------------------------------------
value counts:
        >> more than 50 unique values <<
----------------------------------------------------
mean:1.4e+02, std:1.1e+02, min:0, 25%:9.2, 50%:2.2e+02, 75%:2.4e+02, max:2.4e+02



====================================================
column name = Family_Hist_2, dtype = float64
----------------------------------------
value counts:
```

```
        >> more than 50 unique values <<
--------------------------------------------------
mean:0.47, std:0.15, min:0, 25%:0.36, 50%:0.46, 75%:0.58, max:1



==================================================
column name = Family_Hist_5, dtype = float64
----------------------------------------
value counts:
        >> more than 50 unique values <<
--------------------------------------------------
mean:0.49, std:0.13, min:0, 25%:0.41, 50%:0.51, 75%:0.58, max:1



==================================================
column name = Family_Hist_3, dtype = float64
----------------------------------------
value counts:
        >> more than 50 unique values <<
--------------------------------------------------
mean:0.5, std:0.14, min:0, 25%:0.41, 50%:0.52, 75%:0.61, max:1



==================================================
column name = Medical_History_32, dtype = float64
----------------------------------------
value counts:
        >> more than 50 unique values <<
--------------------------------------------------
mean:12, std:38, min:0, 25%:0, 50%:0, 75%:2, max:2.4e+02



==================================================
column name = Insurance_History_5, dtype = float64
----------------------------------------
value counts:
        >> more than 50 unique values <<
--------------------------------------------------
mean:0.0017, std:0.0065, min:0, 25%:0.0004, 50%:0.00093, 75%:0.002, max:1



==================================================
column name = Medical_History_1, dtype = float64
----------------------------------------
value counts:
        >> more than 50 unique values <<
--------------------------------------------------
mean:7.9, std:13, min:0, 25%:2, 50%:4, 75%:9, max:2.4e+02
```

From the results above, we can see that all of the features that have nulls are continuous features, with values greater than zero. We will go ahead and impute all of the nulls with a value of minus one. (A tree can then easily cut out NaN's separately from the rest of the values.

1 Final imputation

Fill null values with -1 and one-hot encode the `Product_Info_2` feature

```
In [9]: noprod = data.drop(['Product_Info_2'], axis=1)
        dummies = pd.get_dummies(data['Product_Info_2'])

        df = noprod.join(dummies).drop(['Response', 'train?'], axis=1).fillna(-1).join(data[['Response'
        df.to_csv('csvs/data_imputed.csv', index=False)
```

Now we are ready to start training an ensemble

We check that all of the training data indeed falls into one of the 8 categories: