Lab Five: Wide and Deep Networks

Member : Yang Shen

Introduction

Washington, D.C. is the capital of the United States. Washington's population is approaching 700,000 people, and has been growing since 2000 following a half-century of population decline. This dataset contain the posted properties sale information from 2010 to 2018. There are totla 152779 number of records. The null price means that the property has not been sold. I am interested the price of sold properties. I change the price from numeric variable to binary variable. The sell price below 400,000 is normal price and above is high price.

- ID - Unique number for each athlete each event
- BATHRM - Number of full bathrooms
- HEAT_D – heat type
- AC - Y is have AC, N is not
- BEDRM - Number if bed room
- AYB - The earliest time the main portion of the building was built.
- AYB_year - The number of year of earliest time the main portion of the building was built until now
- YR_RMDL - Last year residence was remodeled
- YR_RMDL_year - The number of year until now of last time residence was remodeled
- EYB - The year that an improvement was built that is most often more recent than actual year built.
- EYB_year - The number of year until now of improvement was built
- STORAGE – Number of storage
- PRICE - selling price
- QUALIFIED – Qualified value of 'U' suggests sale is not reflective of market value
- GBA - Gross building area in square feet
- STRUCT_D – Structure description
- CNDTN_D – Condition description
- ROOF_D – Roff discription
- INTWALL_D – Interior wall description
- KITCHENS – Number of kitchens

In [457]:

```
#https://www.kaggle.com/christophercorrea/dc-residential-properties#raw_residential_data.csv
# load the dataset
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')

print('Pandas:', pd.__version__)
print('Numpy:',np.__version__)

dfraw = pd.read_csv('https://raw.githubusercontent.com/sy0701611/lab-3/master/raw_residential_da
ta.csv',error_bad_lines=False)
# read in the csv file,offending lines to be skipped

dfraw.info()
```

```
Pandas: 0.24.2
Numpy: 1.16.2
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 152779 entries, 0 to 152778
Data columns (total 20 columns):
ID              152779 non-null int64
BATHRM          152745 non-null float64
HEAT_D          152745 non-null object
AC              152745 non-null object
BEDRM           152739 non-null float64
AYB             152764 non-null float64
AYB_year        152779 non-null int64
YR_RMDL_year    78419 non-null float64
YR_RMDL         78419 non-null float64
EYB_year        152779 non-null int64
EYB             152779 non-null int64
STORAGE         152669 non-null float64
PRICE           84985 non-null float64
QUALIFIED       152779 non-null object
GBA             152779 non-null int64
STRUCT_D        152745 non-null object
CNDTN_D         152745 non-null object
ROOF_D          152745 non-null object
INTWALL_D       152736 non-null object
KITCHENS        152744 non-null float64
dtypes: float64(8), int64(5), object(7)
memory usage: 23.3+ MB
```

In [458]:

```
# find the duplicate
#no duplicate
idx = dfraw.duplicated()
len(dfraw[idx])
```

Out[458]:

```
0
```

In [459]:

```
#drop na becasue I am interested about the sold property
dfraw.dropna(inplace=True)#drop all na value beacsue row with na is not useful
df = dfraw[['BATHRM','HEAT_D','AC','BEDRM','AYB_year','YR_RMDL_year','EYB_year','STORAGE'
            ,'QUALIFIED','GBA','STRUCT_D','CNDTN_D','ROOF_D','INTWALL_D','KITCHENS','PRICE']]
```

I drop AYB,YR_RMDL,EYB because those are the year, I have variable which will reflect the number of year from their year until now. Which is more meaningful than just the year data. Drop ID becasue ID is not useful.

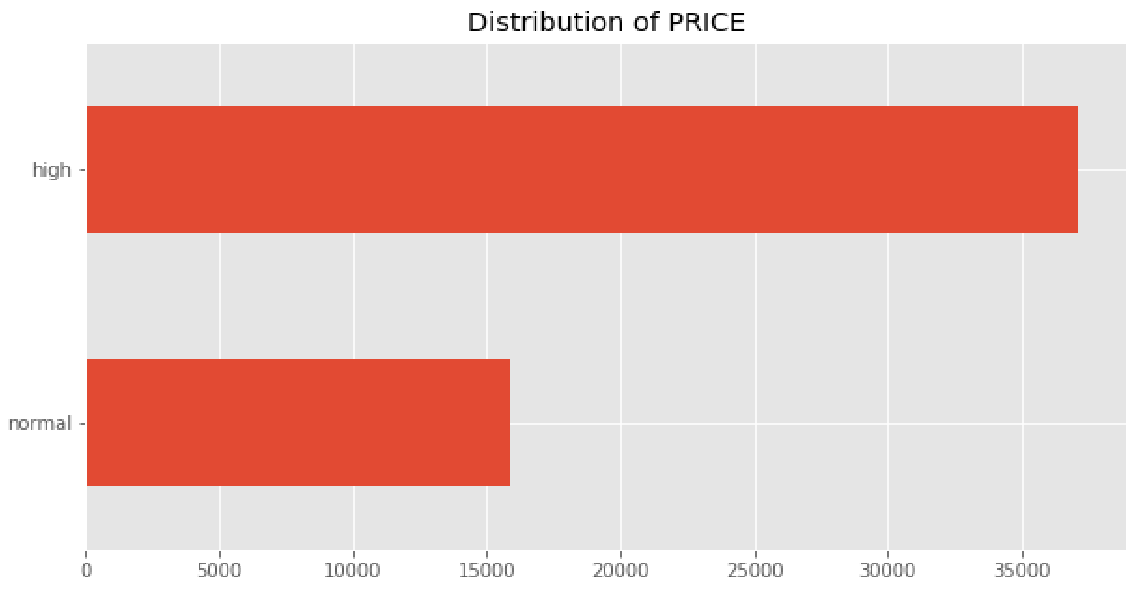
In [460]:

```
df.PRICE = (df.PRICE>400000).astype(np.int)
df.PRICE.replace(to_replace=[0,1],value=['normal','high'],inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 52986 entries, 0 to 152774
Data columns (total 16 columns):
BATHRM          52986 non-null float64
HEAT_D          52986 non-null object
AC              52986 non-null object
BEDRM           52986 non-null float64
AYB_year        52986 non-null int64
YR_RMDL_year    52986 non-null float64
EYB_year        52986 non-null int64
STORAGE         52986 non-null float64
QUALIFIED       52986 non-null object
GBA             52986 non-null int64
STRUCT_D        52986 non-null object
CNDTN_D         52986 non-null object
ROOF_D          52986 non-null object
INTWALL_D       52986 non-null object
KITCHENS        52986 non-null float64
PRICE           52986 non-null object
dtypes: float64(5), int64(3), object(8)
memory usage: 6.9+ MB
```

In [364]:

```
plt.style.use('ggplot')

plt.figure(figsize=(10,5))
df['PRICE'].value_counts(sort=True,ascending=True).plot(kind='barh')
plt.title('Distribution of PRICE')

plt.show()
```



I change the price from numeric variable to binary variable. The sell price below 400,000 is normal price and above is high price.

Dividing training and testing

In [461]:

```
from sklearn.model_selection import train_test_split
df_train_orig, df_test_orig = train_test_split(df, test_size=0.2)
X = df.drop("PRICE", axis=1).as_matrix()
y = df["PRICE"].as_matrix()
```

In [462]:

```
from copy import deepcopy
df_train = deepcopy(df_train_orig)
df_test = deepcopy(df_test_orig)
```

In [463]:

```
import numpy as np

# let's just get rid of rows with any missing data
# and then reset the indices of the dataframe so it corresponds to row number
df_train.replace(to_replace=' ?',value=np.nan, inplace=True)
df_train.dropna(inplace=True)
df_train.reset_index()

df_test.replace(to_replace=' ?',value=np.nan, inplace=True)
df_test.dropna(inplace=True)
df_test.reset_index()

df_test.head()
```

Out[463]:

	BATHRM	HEAT_D	AC	BEDRM	AYB_year	YR_RMDL_year	EYB_year	STORAGE
113826	3.0	Forced Air	Y	4.0	119	3.0	33	2.0
44045	3.0	Forced Air	Y	4.0	97	6.0	37	2.0
20173	1.0	Hot Water Rad	Y	3.0	107	4.0	52	2.0
9217	2.0	Forced Air	Y	4.0	119	47.0	50	2.0
133159	1.0	Forced Air	Y	3.0	109	34.0	62	2.0

I splite my data into training and testing set by 80/20, I did not use any k-fold becasue I have 52000 rows in my dataset after drop the na value, I think 52000 is enough to represent the features of dataset. However, in my comparision of model, I used 5 fold StratifiedKFold, because I need to use cross validation method and also I want my price normal and high have the same ratio in each fold. I used both 80/20 and StratifiedKFold as I need.

In [464]:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

#replace strings
#if df_test.PRICE.dtype=='object':
#    df_test.PRICE.replace(to_replace=['0','1'],value=['normal','high'],inplace=True)
#    print(df_test.income.value_counts())

# define objects that can encode each variable as integer
encoders = dict()
categorical_headers = ['HEAT_D','AC','QUALIFIED','STRUCT_D','CNDTN_D','ROOF_D','INTWALL_D']

# train all encoders (special case the target 'PRICE')
for col in categorical_headers+['PRICE']:
    df_train[col] = df_train[col].str.strip()
    df_test[col] = df_test[col].str.strip()

    if col=="PRICE":
        tmp = LabelEncoder()
        df_train[col] = tmp.fit_transform(df_train[col])
        df_test[col] = tmp.transform(df_test[col])
    else:
        # integer encoded variables
        encoders[col] = LabelEncoder() # save the encoder
        df_train[col+'_int'] = encoders[col].fit_transform(df_train[col])
        df_test[col+'_int'] = encoders[col].transform(df_test[col])

# scale the numeric, continuous variables
numeric_headers = ["BATHRM","BEDRM","AYB_year","YR_RMDL_year","EYB_year","STORAGE"
            ,"GBA","KITCHENS"]

for col in numeric_headers:
    df_train[col] = df_train[col].astype(np.float)
    df_test[col] = df_test[col].astype(np.float)

    ss = StandardScaler()
    df_train[col] = ss.fit_transform(df_train[col].values.reshape(-1, 1))
    df_test[col] = ss.transform(df_test[col].values.reshape(-1, 1))

df_train.head()
```

Out[464]:

	BATHRM	HEAT_D	AC	BEDRM	AYB_year	YR_RMDL_year	EYB_year	STORAGE
32405	-1.273703	Forced Air	Y	-0.359388	-0.311925	-0.028223	0.614581	-0.023048
143276	-0.305399	Forced Air	Y	-0.359388	-0.134518	-0.239958	1.195303	-0.023048
34935	1.631208	Forced Air	Y	-0.359388	-0.543919	-0.187024	-0.353288	-0.023048
141575	-0.305399	Warm Cool	Y	-0.359388	0.097475	-0.398759	0.033860	-0.023048
102427	-0.305399	Warm Cool	Y	-0.359388	-0.407452	-0.663428	1.582451	-0.023048

5 rows × 23 columns

Normalized

I used standard scalar to normalized my numeric variables. The idea behind StandardScaler is that it will transform my data such that its distribution will have a mean value 0 and standard deviation of 1. So that all numeric values are in same scales. no more really big data or really small data.

In [465]:

```
# let's start as simply as possible, without any feature preprocessing
categorical_headers_ints = [x+'_int' for x in categorical_headers]

# we will forego one-hot encoding right now and instead just scale all inputs
#   this is just to get an example running in Keras (don't ever do this)
feature_columns = categorical_headers_ints+numeric_headers
X_train =  ss.fit_transform(df_train[feature_columns].values).astype(np.float32)
X_test =  ss.transform(df_test[feature_columns].values).astype(np.float32)

y_train = df_train['PRICE'].values.astype(np.int)
y_test = df_test['PRICE'].values.astype(np.int)

print(feature_columns)
```

```
['HEAT_D_int', 'AC_int', 'QUALIFIED_int', 'STRUCT_D_int', 'CNDTN_D_int', 'ROOF_D_i
nt', 'INTWALL_D_int', 'BATHRM', 'BEDRM', 'AYB_year', 'YR_RMDL_year', 'EYB_year',
'STORAGE', 'GBA', 'KITCHENS']
```

In [466]:

```
from sklearn import metrics as mt
import keras

keras.__version__
```

Out[466]:

```
'2.2.4'
```

In [467]:

```
# from keras.models import Sequential
from keras.layers import Dense, Activation, Input
from keras.layers import Embedding, Flatten, Concatenate
from keras.models import Model
import warnings
warnings.filterwarnings('ignore')
```

Metrics

For my dataset, I have two metrics which is accuracy and FN score. I used accuracy because accuracy will directly reflect how well my prediction is. I want my model have high true positive rate and high true negative rate. Which means my model can predict the price well.

Another metric I am using here is customized FN score. It is calculated by FN/(FN+FP). For my case, if it predict false positive, that is ok because if we set low price house to a high price. The consequance is that nobody buy it. It just wasted of time, but no real money loss. However, false negative is we set high price house to low price to sell. That will cause seller loss money. They will not sell as their house should be. So that we want the false negative rate as low as possible. My FN score will reflect the ratio of false negative and all false prediction.

In [468]:

```
#This is my own metric which is the FN/(FN+FP)
def custom_score(y_true, y_predict):
    cm = mt.confusion_matrix(y_test,yhat)
    FN = cm[1,0]/(cm[0,1]+cm[1,0])

    return FN
```

Cross-product features

HEAT_D and AC: for the most house, they have both ac and heat, I combined these two because heat and AC is really important for house. They are similar, both of them can control temperature.

QUALIFIED and CNDTN_D: I think condition of house will reflect the qualified. Some house have good condition, so they sell reflect market value. I think those two are related.

ROOF_D and INTWALL_D: I think the material of roof and wall are related really closed. For example, I hot place, roof is usually reflect hear and wall is isolated heat from outside. Those materials are paired to perform the same function.

HEAT_D and INTWALL_D: For some hot area, they do not have heat or it is combine with AC, the wall meterial is also have same function to control the temperature as heat.

AC and ROOF_D: Same story as heat and wall, roof can reflect heat from sun. It have same function as AC, which is make house cool, So I think those two are meaningful to cross.

ROOF_D , INTWALL_D and STRUCT_D: Those are meterial for house to build, I think cross product them are meaningful. It will reflect the house construction.

Model

Model1 with 3 layer in Deep and cross combo1

In [423]:

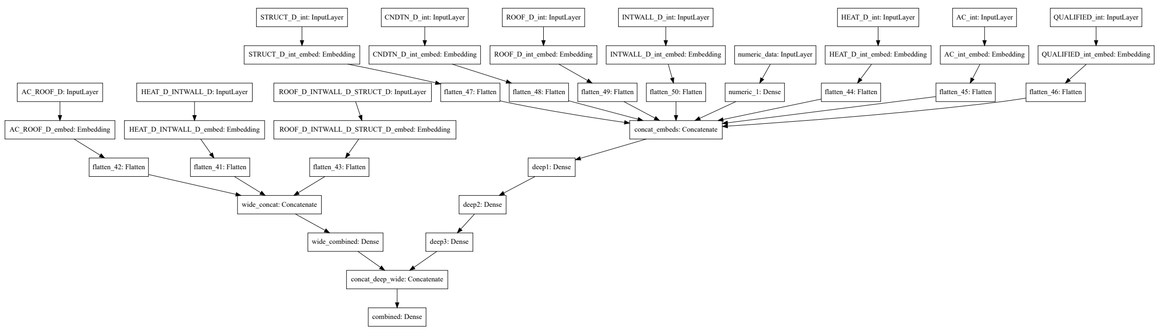
```
#model1
from keras.layers import concatenate
from sklearn.preprocessing import OneHotEncoder


#'HEAT_D','AC','QUALIFIED','STRUCT_D','CNDTN_D','ROOF_D','INTWALL_D'
#"BATHRM","BEDRM","AYB_year","YR_RMDL_year","EYB_year","STORAGE","GBA","KITCHENS"
cross_columns1 = [['HEAT_D','AC'],
                 ['QUALIFIED', 'CNDTN_D'],
                 ['ROOF_D','INTWALL_D']]

# and save off the numeric features
X_train_num =  df_train[numeric_headers].values
X_test_num = df_test[numeric_headers].values
#def create_model this just add in frount of original Dr. Larson code, doing this for cross vali
dation
#https://github.com/Thakugan/machine-learning-notebooks/blob/master/6-wide-and-deep-networks/mus
hroom-hunting.ipynb
def create_model(X_train, X_test, X_train_num, categorical_headers_ints, cross_columns=cross_col
umns1):
    #this line is the only different campare to Dr. Larson code, this def is used for cross vali
dation for loop
    embed_branches = []
    X_ints_train = []
    X_ints_test = []
    all_inputs = []
    all_wide_branch_outputs = []

    for cols in cross_columns:
    # encode crossed columns as ints for the embedding
        enc = LabelEncoder()

    # create crossed labels
        X_crossed_train = df_train[cols].apply(lambda x: '_'.join(x), axis=1)
        X_crossed_test = df_test[cols].apply(lambda x: '_'.join(x), axis=1)

        enc.fit(np.hstack((X_crossed_train.values,  X_crossed_test.values)))
        X_crossed_train = enc.transform(X_crossed_train)
        X_crossed_test = enc.transform(X_crossed_test)
        X_ints_train.append( X_crossed_train )
        X_ints_test.append( X_crossed_test )

    # get the number of categories
        N = max(X_ints_train[-1]+1) # same as the max(df_train[col])

    # create embedding branch from the number of categories
        inputs = Input(shape=(1,),dtype='int32', name = '_'.join(cols))
        all_inputs.append(inputs)
        x = Embedding(input_dim=N,
                  output_dim=int(np.sqrt(N)),
                  input_length=1, name = '_'.join(cols)+'_embed')(inputs)
        x = Flatten()(x)
        all_wide_branch_outputs.append(x)

# merge the branches together
    wide_branch = concatenate(all_wide_branch_outputs, name='wide_concat')
    wide_branch = Dense(units=1,activation='sigmoid',name='wide_combined')(wide_branch)

# reset this input branch
```

```
    all_deep_branch_outputs = []
# add in the embeddings
    for col in categorical_headers_ints:
    # encode as ints for the embedding
        X_ints_train.append( df_train[col].values )
        X_ints_test.append( df_test[col].values )

    # get the number of categories
        N = max(X_ints_train[-1]+1) # same as the max(df_train[col])

    # create embedding branch from the number of categories
        inputs = Input(shape=(1,),dtype='int32', name=col)
        all_inputs.append(inputs)
        x = Embedding(input_dim=N,
                    output_dim=int(np.sqrt(N)),
                    input_length=1, name=col+'_embed')(inputs)
        x = Flatten()(x)
        all_deep_branch_outputs.append(x)

# also get a dense branch of the numeric features
    all_inputs.append(Input(shape=(X_train_num.shape[1],),
                          sparse=False,
                          name='numeric_data'))

    x = Dense(units=20, activation='relu',name='numeric_1')(all_inputs[-1])
    all_deep_branch_outputs.append( x )

# merge the deep branches together
    deep_branch = concatenate(all_deep_branch_outputs,name='concat_embeds')
    deep_branch = Dense(units=50,activation='relu', name='deep1')(deep_branch)
    deep_branch = Dense(units=25,activation='relu', name='deep2')(deep_branch)
    deep_branch = Dense(units=10,activation='relu', name='deep3')(deep_branch)

    final_branch = concatenate([wide_branch, deep_branch],name='concat_deep_wide')
    final_branch = Dense(units=1,activation='sigmoid',name='combined')(final_branch)

    model = Model(inputs=all_inputs, outputs=final_branch)
    return model, X_ints_train, X_ints_test, X_train_num
```

In [424]:

```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

# you will need to install pydot properly on your machine to get this running
SVG(model_to_dot(model).create(prog='dot', format='svg'))
```

Out[424]:



In [425]:

```
model.compile(optimizer='adagrad',
              loss='mean_squared_error',
              metrics=['accuracy'])
```

In [426]:

```
# lets also add the history variable to see how we are doing
# and lets add a validation set to keep track of our progress
history = model.fit(X_ints_train+ [X_train_num],
                    y_train,
                    epochs=30,
                    batch_size=50,
                    verbose=1,
                    validation_data = (X_ints_test + [X_test_num], y_test))
```

```
Train on 42388 samples, validate on 10598 samples
Epoch 1/30
42388/42388 [==============================] - 6s 133us/step - loss: 0.1172 - acc:
0.8365 - val_loss: 0.1210 - val_acc: 0.8303
Epoch 2/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1132 - acc:
0.8439 - val_loss: 0.1218 - val_acc: 0.8269
Epoch 3/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1125 - acc:
0.8451 - val_loss: 0.1235 - val_acc: 0.8257
Epoch 4/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1119 - acc:
0.8465 - val_loss: 0.1205 - val_acc: 0.8311
Epoch 5/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1115 - acc:
0.8470 - val_loss: 0.1210 - val_acc: 0.8297
Epoch 6/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1111 - acc:
0.8480 - val_loss: 0.1202 - val_acc: 0.8326
Epoch 7/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1109 - acc:
0.8484 - val_loss: 0.1206 - val_acc: 0.8315
Epoch 8/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1106 - acc:
0.8495 - val_loss: 0.1205 - val_acc: 0.8303
Epoch 9/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1105 - acc:
0.8490 - val_loss: 0.1204 - val_acc: 0.8311
Epoch 10/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1102 - acc:
0.8497 - val_loss: 0.1203 - val_acc: 0.8315
Epoch 11/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1101 - acc:
0.8492 - val_loss: 0.1206 - val_acc: 0.8314
Epoch 12/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1099 - acc:
0.8505 - val_loss: 0.1203 - val_acc: 0.8314
Epoch 13/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1098 - acc:
0.8499 - val_loss: 0.1203 - val_acc: 0.8307
Epoch 14/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1096 - acc:
0.8509 - val_loss: 0.1203 - val_acc: 0.8306
Epoch 15/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1095 - acc:
0.8507 - val_loss: 0.1202 - val_acc: 0.8313
Epoch 16/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1094 - acc:
0.8508 - val_loss: 0.1208 - val_acc: 0.8306
Epoch 17/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1092 - acc:
0.8516 - val_loss: 0.1204 - val_acc: 0.8315
Epoch 18/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1091 - acc:
0.8521 - val_loss: 0.1201 - val_acc: 0.8335
Epoch 19/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1090 - acc:
0.8524 - val_loss: 0.1205 - val_acc: 0.8322
Epoch 20/30
42388/42388 [==============================] - 5s 112us/step - loss: 0.1089 - acc:
0.8521 - val_loss: 0.1204 - val_acc: 0.8320
```

```
Epoch 21/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1088 - acc: 
0.8526 - val_loss: 0.1205 - val_acc: 0.8322
Epoch 22/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1087 - acc: 
0.8531 - val_loss: 0.1202 - val_acc: 0.8317
Epoch 23/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1086 - acc: 
0.8533 - val_loss: 0.1203 - val_acc: 0.8319
Epoch 24/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1085 - acc: 
0.8540 - val_loss: 0.1204 - val_acc: 0.8325
Epoch 25/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1084 - acc: 
0.8526 - val_loss: 0.1198 - val_acc: 0.8336
Epoch 26/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1084 - acc: 
0.8537 - val_loss: 0.1200 - val_acc: 0.8334
Epoch 27/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1083 - acc: 
0.8541 - val_loss: 0.1201 - val_acc: 0.8332
Epoch 28/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1082 - acc: 
0.8543 - val_loss: 0.1202 - val_acc: 0.8325
Epoch 29/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1081 - acc: 
0.8545 - val_loss: 0.1202 - val_acc: 0.8319
Epoch 30/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1080 - acc: 
0.8548 - val_loss: 0.1202 - val_acc: 0.8322
```

In [402]:

```
#accuracy score
yhat = np.round(model.predict(X_ints_test + [X_test_num]))
print(mt.confusion_matrix(y_test,yhat), 'Accuracy ',mt.accuracy_score(y_test,yhat))
#my FN score
print('FN score ',custom_score(y_test,yhat))
```

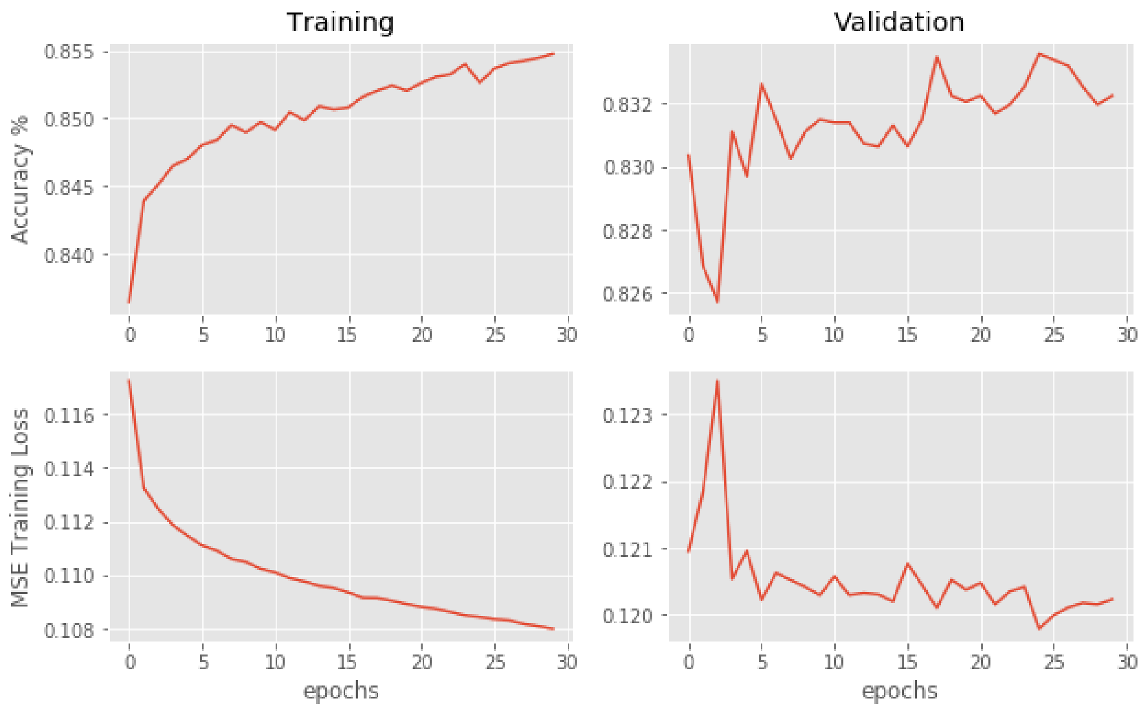
```
[[6646  731]
 [1139 2082]] Accuracy  0.8235516135119834
FN score  0.6090909090909091
```

In [427]:

```
from matplotlib import pyplot as plt

%matplotlib inline

plt.figure(figsize=(10,6))
plt.subplot(2,2,1)
plt.plot(history.history['acc'])

plt.ylabel('Accuracy %')
plt.title('Training')
plt.subplot(2,2,2)
plt.plot(history.history['val_acc'])
plt.title('Validation')

plt.subplot(2,2,3)
plt.plot(history.history['loss'])
plt.ylabel('MSE Training Loss')
plt.xlabel('epochs')

plt.subplot(2,2,4)
plt.plot(history.history['val_loss'])
plt.xlabel('epochs')
```

Out[427]:

```
Text(0.5, 0, 'epochs')
```



For train vs loss, I think it converge, the line reach the bottom. For validation, the line is boucing after 5 epochs, may be stop at 20 epochs is ok. But in general, they converge.

In [441]:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html#
sklearn.model_selection.StratifiedKFold.split
#https://github.com/Thakugan/machine-learning-notebooks/blob/master/6-wide-and-deep-networks/mus
hroom-hunting.ipynb
num_folds = 5

acc_scores =[]
fn_scores = []

skf = StratifiedKFold(n_splits=num_folds, shuffle=True)
for i, (train, test) in enumerate(skf.split(X, y)):#here I used corss validation on my model

    model, X_ints_train, X_ints_test, X_train_num = create_model(X_train, X_test, X_train_num, c
ategorical_headers_ints, cross_columns1)
    #doing modeling same as above, without for loop
    model.compile(optimizer='adagrad',
                  loss='mean_squared_error',
                  metrics=['accuracy'])

    model.fit(X_ints_train + [X_train_num], y_train, epochs=30, batch_size=50, verbose=1)
#this is just what i do without cross validation
    yhat = np.round(model.predict(X_ints_test + [X_test_num]))
    print(mt.confusion_matrix(y_test,yhat))
    acc_score = mt.accuracy_score(y_test, yhat)
    acc_scores.append(acc_score)#append all acc for k-fold
    fn_score = custom_score(y_test,yhat)
    fn_scores.append(fn_score)#append all fn for k-fold
    print("Accuracy: ", acc_score)
    print("Fn : ", fn_score)
print(fn_scores)
print(acc_scores)
```

```
Epoch 1/30
42388/42388 [==============================] - 6s 147us/step - loss: 0.1363 - acc:
0.8050
Epoch 2/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1302 - acc:
0.8146
Epoch 3/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1290 - acc:
0.8169
Epoch 4/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1284 - acc:
0.8177
Epoch 5/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1279 - acc:
0.8187
Epoch 6/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1275 - acc:
0.8194
Epoch 7/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1271 - acc:
0.8203
Epoch 8/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1269 - acc:
0.8211
Epoch 9/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1266 - acc:
0.8209
Epoch 10/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1264 - acc:
0.8212
Epoch 11/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1261 - acc:
0.8215
Epoch 12/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1260 - acc:
0.8217
Epoch 13/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1258 - acc:
0.8233
Epoch 14/30
42388/42388 [==============================] - 4s 103us/step - loss: 0.1255 - acc:
0.8226
Epoch 15/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1254 - acc:
0.8235
Epoch 16/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1252 - acc:
0.8232
Epoch 17/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1251 - acc:
0.8236
Epoch 18/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1249 - acc:
0.8239
Epoch 19/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1248 - acc:
0.8242
Epoch 20/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1246 - acc:
0.8242
Epoch 21/30
```

```
42388/42388 [==============================] - 4s 105us/step - loss: 0.1245 - acc:
0.8241
Epoch 22/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1243 - acc:
0.8247
Epoch 23/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1241 - acc:
0.8252
Epoch 24/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1241 - acc:
0.8252
Epoch 25/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1239 - acc:
0.8258
Epoch 26/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1238 - acc:
0.8259
Epoch 27/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1237 - acc:
0.8260
Epoch 28/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1235 - acc:
0.8262
Epoch 29/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1234 - acc:
0.8257
Epoch 30/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1233 - acc:
0.8263
[[6584  793]
 [1102 2119]]
Accuracy:  0.8211926778637478
Fn :  0.5815303430079156
Epoch 1/30
42388/42388 [==============================] - 6s 145us/step - loss: 0.1368 - acc:
0.8053
Epoch 2/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1301 - acc:
0.8155
Epoch 3/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1291 - acc:
0.8177
Epoch 4/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1284 - acc:
0.8189
Epoch 5/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1278 - acc:
0.8194
Epoch 6/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1274 - acc:
0.8201
Epoch 7/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1271 - acc:
0.8190
Epoch 8/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1267 - acc:
0.8208
Epoch 9/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1264 - acc:
0.8213
Epoch 10/30
```

```
42388/42388 [==============================] - 5s 108us/step - loss: 0.1261 - acc:
0.8210
Epoch 11/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1259 - acc:
0.8218
Epoch 12/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1256 - acc:
0.8225
Epoch 13/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1254 - acc:
0.8231
Epoch 14/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1252 - acc:
0.8223
Epoch 15/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1250 - acc:
0.8235
Epoch 16/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1248 - acc:
0.8232
Epoch 17/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1246 - acc:
0.8240
Epoch 18/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1244 - acc:
0.8248
Epoch 19/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1243 - acc:
0.8244
Epoch 20/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1241 - acc:
0.8257
Epoch 21/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1239 - acc:
0.8253
Epoch 22/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1238 - acc:
0.8257
Epoch 23/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1236 - acc:
0.8262
Epoch 24/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1235 - acc:
0.8262
Epoch 25/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1233 - acc:
0.8265
Epoch 26/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1232 - acc:
0.8264
Epoch 27/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1230 - acc:
0.8271
Epoch 28/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1229 - acc:
0.8267
Epoch 29/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1228 - acc:
0.8275
Epoch 30/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1227 - acc:
```

```
0.8271
[[6650  727]
 [1115 2106]]
Accuracy:  0.8261936214380071
Fn :  0.6053203040173725
Epoch 1/30
42388/42388 [==============================] - 6s 150us/step - loss: 0.1360 - acc:
0.8061
Epoch 2/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1295 - acc:
0.8160
Epoch 3/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1284 - acc:
0.8177
Epoch 4/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1275 - acc:
0.8197
Epoch 5/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1270 - acc:
0.8190
Epoch 6/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1266 - acc:
0.8204
Epoch 7/30
42388/42388 [==============================] - 5s 114us/step - loss: 0.1262 - acc:
0.8211
Epoch 8/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1259 - acc:
0.8213
Epoch 9/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1256 - acc:
0.8216
Epoch 10/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1253 - acc:
0.8224
Epoch 11/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1251 - acc:
0.8227
Epoch 12/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1249 - acc:
0.8236
Epoch 13/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1246 - acc:
0.8231
Epoch 14/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1244 - acc:
0.8235
Epoch 15/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1242 - acc:
0.8240
Epoch 16/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1241 - acc:
0.8246
Epoch 17/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1239 - acc:
0.8254
Epoch 18/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1237 - acc:
0.8255
Epoch 19/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1235 - acc:
```

```
0.8258
Epoch 20/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1233 - acc:
0.8265
Epoch 21/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1231 - acc:
0.8258
Epoch 22/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1230 - acc:
0.8262
Epoch 23/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1228 - acc:
0.8264
Epoch 24/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1227 - acc:
0.8268
Epoch 25/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1225 - acc:
0.8274
Epoch 26/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1224 - acc:
0.8262
Epoch 27/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1223 - acc:
0.8276
Epoch 28/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1221 - acc:
0.8276
Epoch 29/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1220 - acc:
0.8279
Epoch 30/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1218 - acc:
0.8280
[[6589  788]
 [1068 2153]]
Accuracy:  0.8248726174749953
Fn :  0.5754310344827587
Epoch 1/30
42388/42388 [==============================] - 6s 152us/step - loss: 0.1367 - acc:
0.8060
Epoch 2/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1302 - acc:
0.8154
Epoch 3/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1294 - acc:
0.8166
Epoch 4/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1289 - acc:
0.8174
Epoch 5/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1284 - acc:
0.8172
Epoch 6/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1280 - acc:
0.8186
Epoch 7/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1277 - acc:
0.8183
Epoch 8/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1274 - acc:
```

```
0.8190
Epoch 9/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1272 - acc:
0.8195
Epoch 10/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1269 - acc:
0.8193
Epoch 11/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1266 - acc:
0.8201
Epoch 12/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1264 - acc:
0.8197
Epoch 13/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1262 - acc:
0.8210
Epoch 14/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1260 - acc:
0.8205
Epoch 15/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1259 - acc:
0.8210
Epoch 16/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1257 - acc:
0.8217
Epoch 17/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1255 - acc:
0.8219
Epoch 18/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1253 - acc:
0.8220
Epoch 19/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1251 - acc:
0.8224
Epoch 20/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1250 - acc:
0.8223
Epoch 21/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1249 - acc:
0.8220
Epoch 22/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1247 - acc:
0.8236
Epoch 23/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1246 - acc:
0.8234
Epoch 24/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1245 - acc:
0.8231
Epoch 25/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1243 - acc:
0.8237
Epoch 26/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1241 - acc:
0.8237
Epoch 27/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1240 - acc:
0.8243
Epoch 28/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1239 - acc:
0.8243
```

```
Epoch 29/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1237 - acc:
0.8243
Epoch 30/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1235 - acc:
0.8249
[[6644  733]
 [1176 2045]]
Accuracy:  0.819871673900736
Fn :  0.6160293347302253
Epoch 1/30
42388/42388 [==============================] - 7s 154us/step - loss: 0.1379 - acc:
0.8035
Epoch 2/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1306 - acc:
0.8156
Epoch 3/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1295 - acc:
0.8179
Epoch 4/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1288 - acc:
0.8190
Epoch 5/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1283 - acc:
0.8189
Epoch 6/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1278 - acc:
0.8197
Epoch 7/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1274 - acc:
0.8206
Epoch 8/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1270 - acc:
0.8209
Epoch 9/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1267 - acc:
0.8210
Epoch 10/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1265 - acc:
0.8213
Epoch 11/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1261 - acc:
0.8222
Epoch 12/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1260 - acc:
0.8229
Epoch 13/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1256 - acc:
0.8219
Epoch 14/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1255 - acc:
0.8237
Epoch 15/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1253 - acc:
0.8231
Epoch 16/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1250 - acc:
0.8236
Epoch 17/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1249 - acc:
0.8245
```

```
Epoch 18/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1247 - acc:
0.8247
Epoch 19/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1245 - acc:
0.8243
Epoch 20/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1243 - acc:
0.8250
Epoch 21/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1242 - acc:
0.8247
Epoch 22/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1240 - acc:
0.8248
Epoch 23/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1239 - acc:
0.8258
Epoch 24/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1237 - acc:
0.8260
Epoch 25/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1236 - acc:
0.8258
Epoch 26/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1234 - acc:
0.8258
Epoch 27/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1233 - acc:
0.8258
Epoch 28/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1231 - acc:
0.8267
Epoch 29/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1230 - acc:
0.8271
Epoch 30/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1229 - acc:
0.8274
[[6669  708]
 [1154 2067]]
Accuracy:  0.8243064729194187
Fn :  0.6197636949516648
[0.5815303430079156, 0.6053203040173725, 0.5754310344827587, 0.6160293347302253,
0.6197636949516648]
[0.8211926778637478, 0.8261936214380071, 0.8248726174749953, 0.819871673900736, 0.
8243064729194187]
```

Second model

Model2 with 3 layer in Deep and cross combo2

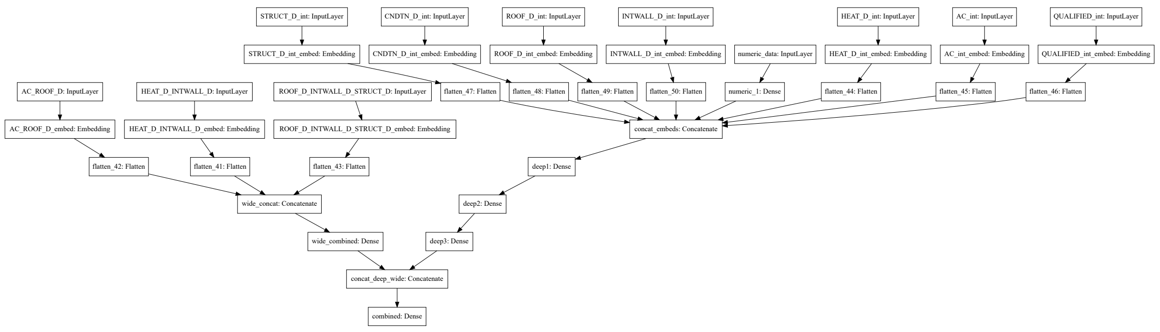
In [443]:

```
from keras.layers import concatenate
from sklearn.preprocessing import OneHotEncoder
#model2
#'HEAT_D','AC','QUALIFIED','STRUCT_D','CNDTN_D','ROOF_D','INTWALL_D'
#"BATHRM", "BEDRM", "AYB_year", "YR_RMDL_year", "EYB_year", "STORAGE", "GBA", "KITCHENS"
cross_columns2 = [['HEAT_D','INTWALL_D'],
                  ['AC', 'ROOF_D'],
                  ['ROOF_D','INTWALL_D','STRUCT_D']]



# and save off the numeric features
X_train_num =  df_train[numeric_headers].values
X_test_num = df_test[numeric_headers].values

#this def model is prepare for the cross validation
def create_model1(X_train, X_test, X_train_num, categorical_headers_ints, cross_columns=cross_co
lumns2):
    embed_branches = []
    X_ints_train = []
    X_ints_test = []
    all_inputs = []
    all_wide_branch_outputs = []

    for cols in cross_columns:
    # encode crossed columns as ints for the embedding
        enc = LabelEncoder()

    # create crossed labels
        X_crossed_train = df_train[cols].apply(lambda x: '_'.join(x), axis=1)
        X_crossed_test = df_test[cols].apply(lambda x: '_'.join(x), axis=1)

        enc.fit(np.hstack((X_crossed_train.values,  X_crossed_test.values)))
        X_crossed_train = enc.transform(X_crossed_train)
        X_crossed_test = enc.transform(X_crossed_test)
        X_ints_train.append( X_crossed_train )
        X_ints_test.append( X_crossed_test )

    # get the number of categories
        N = max(X_ints_train[-1]+1) # same as the max(df_train[col])

    # create embedding branch from the number of categories
        inputs = Input(shape=(1,),dtype='int32', name = '_'.join(cols))
        all_inputs.append(inputs)
        x = Embedding(input_dim=N,
                      output_dim=int(np.sqrt(N)),
                      input_length=1, name = '_'.join(cols)+'_embed')(inputs)
        x = Flatten()(x)
        all_wide_branch_outputs.append(x)

# merge the branches together
    wide_branch = concatenate(all_wide_branch_outputs, name='wide_concat')
    wide_branch = Dense(units=1,activation='sigmoid',name='wide_combined')(wide_branch)

# reset this input branch
    all_deep_branch_outputs = []
# add in the embeddings
    for col in categorical_headers_ints:
    # encode as ints for the embedding
        X_ints_train.append( df_train[col].values )
        X_ints_test.append( df_test[col].values )

    # get the number of categories
        N = max(X_ints_train[-1]+1) # same as the max(df_train[col])

    # create embedding branch from the number of categories
        inputs = Input(shape=(1,),dtype='int32', name=col)
        all_inputs.append(inputs)
        x = Embedding(input_dim=N,
                  output_dim=int(np.sqrt(N)),
                  input_length=1, name=col+'_embed')(inputs)
        x = Flatten()(x)
        all_deep_branch_outputs.append(x)

# also get a dense branch of the numeric features
    all_inputs.append(Input(shape=(X_train_num.shape[1],),
                            sparse=False,
                            name='numeric_data'))

    x = Dense(units=20, activation='relu',name='numeric_1')(all_inputs[-1])
    all_deep_branch_outputs.append( x )

# merge the deep branches together
    deep_branch = concatenate(all_deep_branch_outputs,name='concat_embeds')
    deep_branch = Dense(units=50,activation='relu', name='deep1')(deep_branch)
    deep_branch = Dense(units=25,activation='relu', name='deep2')(deep_branch)
    deep_branch = Dense(units=10,activation='relu', name='deep3')(deep_branch)

    final_branch = concatenate([wide_branch, deep_branch],name='concat_deep_wide')
    final_branch = Dense(units=1,activation='sigmoid',name='combined')(final_branch)

    model1 = Model(inputs=all_inputs, outputs=final_branch)
    return model1, X_ints_train, X_ints_test, X_train_num
```

In [361]:

```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

# you will need to install pydot properly on your machine to get this running
SVG(model_to_dot(model).create(prog='dot', format='svg'))
```

Out[361]:



In [372]:

```
model1.compile(optimizer='adagrad',
              loss='mean_squared_error',
              metrics=['accuracy'])
```

In [373]:

```
# lets also add the history variable to see how we are doing
# and lets add a validation set to keep track of our progress
history1 = model1.fit(X_ints_train+ [X_train_num],
                    y_train,
                    epochs=30,
                    batch_size=50,
                    verbose=1,
                    validation_data = (X_ints_test + [X_test_num], y_test))
```

```
Train on 42388 samples, validate on 10598 samples
Epoch 1/30
42388/42388 [==============================] - 5s 124us/step - loss: 0.1410 - acc:
0.7989 - val_loss: 0.1304 - val_acc: 0.8148
Epoch 2/30
42388/42388 [==============================] - 4s 103us/step - loss: 0.1300 - acc:
0.8153 - val_loss: 0.1284 - val_acc: 0.8168
Epoch 3/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1286 - acc:
0.8174 - val_loss: 0.1273 - val_acc: 0.8182
Epoch 4/30
42388/42388 [==============================] - 4s 103us/step - loss: 0.1277 - acc:
0.8181 - val_loss: 0.1270 - val_acc: 0.8188
Epoch 5/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1272 - acc:
0.8188 - val_loss: 0.1270 - val_acc: 0.8193
Epoch 6/30
42388/42388 [==============================] - 4s 103us/step - loss: 0.1267 - acc:
0.8192 - val_loss: 0.1268 - val_acc: 0.8177
Epoch 7/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1263 - acc:
0.8195 - val_loss: 0.1260 - val_acc: 0.8197
Epoch 8/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1260 - acc:
0.8199 - val_loss: 0.1258 - val_acc: 0.8199
Epoch 9/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1256 - acc:
0.8205 - val_loss: 0.1257 - val_acc: 0.8209
Epoch 10/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1253 - acc:
0.8211 - val_loss: 0.1255 - val_acc: 0.8204
Epoch 11/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1252 - acc:
0.8220 - val_loss: 0.1259 - val_acc: 0.8206
Epoch 12/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1250 - acc:
0.8223 - val_loss: 0.1252 - val_acc: 0.8212
Epoch 13/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1248 - acc:
0.8223 - val_loss: 0.1253 - val_acc: 0.8215
Epoch 14/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1246 - acc:
0.8229 - val_loss: 0.1252 - val_acc: 0.8203
Epoch 15/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1244 - acc:
0.8230 - val_loss: 0.1250 - val_acc: 0.8207
Epoch 16/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1242 - acc:
0.8237 - val_loss: 0.1248 - val_acc: 0.8214
Epoch 17/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1241 - acc:
0.8234 - val_loss: 0.1251 - val_acc: 0.8209
Epoch 18/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1239 - acc:
0.8242 - val_loss: 0.1248 - val_acc: 0.8219
Epoch 19/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1238 - acc:
0.8238 - val_loss: 0.1245 - val_acc: 0.8227
Epoch 20/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1236 - acc:
0.8249 - val_loss: 0.1247 - val_acc: 0.8222
```

```
Epoch 21/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1235 - acc:
0.8247 - val_loss: 0.1246 - val_acc: 0.8221
Epoch 22/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1234 - acc:
0.8245 - val_loss: 0.1244 - val_acc: 0.8228
Epoch 23/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1233 - acc:
0.8248 - val_loss: 0.1245 - val_acc: 0.8223
Epoch 24/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1232 - acc:
0.8249 - val_loss: 0.1243 - val_acc: 0.8219
Epoch 25/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1231 - acc:
0.8250 - val_loss: 0.1244 - val_acc: 0.8219
Epoch 26/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1230 - acc:
0.8250 - val_loss: 0.1243 - val_acc: 0.8218
Epoch 27/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1228 - acc:
0.8255 - val_loss: 0.1243 - val_acc: 0.8221
Epoch 28/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1227 - acc:
0.8255 - val_loss: 0.1243 - val_acc: 0.8211
Epoch 29/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1226 - acc:
0.8267 - val_loss: 0.1241 - val_acc: 0.8217
Epoch 30/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1226 - acc:
0.8263 - val_loss: 0.1241 - val_acc: 0.8222
```

In [403]:

```
#accuracy score
yhat = np.round(model1.predict(X_ints_test + [X_test_num]))
print(mt.confusion_matrix(y_test,yhat), 'Accuracy ',mt.accuracy_score(y_test,yhat))
#my FN score
print('FN score ',custom_score(y_test,yhat))
```

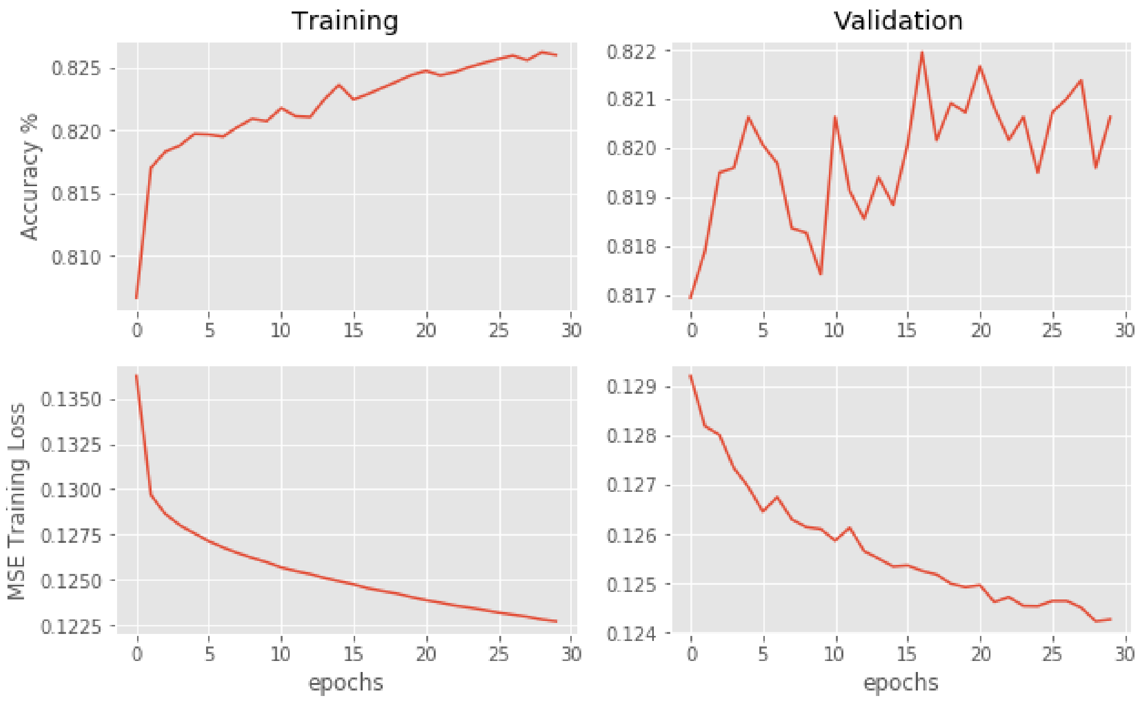
```
[[6602  775]
 [1087 2134]] Accuracy  0.8243064729194187
FN score  0.583780880773362
```

In [366]:

```
from matplotlib import pyplot as plt

%matplotlib inline

plt.figure(figsize=(10,6))
plt.subplot(2,2,1)
plt.plot(history1.history['acc'])

plt.ylabel('Accuracy %')
plt.title('Training')
plt.subplot(2,2,2)
plt.plot(history1.history['val_acc'])
plt.title('Validation')

plt.subplot(2,2,3)
plt.plot(history1.history['loss'])
plt.ylabel('MSE Training Loss')
plt.xlabel('epochs')

plt.subplot(2,2,4)
plt.plot(history1.history['val_loss'])
plt.xlabel('epochs')
```

Out[366]:

```
Text(0.5, 0, 'epochs')
```



For train vs loss, I think it converge, the line reach the bottom. For validation, the line boucing around but keep going down. In general, they converge. i think 30 epochs is fine.

In [444]:

```
#https://github.com/Thakugan/machine-learning-notebooks/blob/master/6-wide-and-deep-networks/mus
hroom-hunting.ipynb
num_folds = 5

acc_scores1 =[]
fn_scores1 = []

skf = StratifiedKFold(n_splits=num_folds, shuffle=True)
for i, (train, test) in enumerate(skf.split(X, y)):#here I used corss validation on my model

    model1, X_ints_train, X_ints_test, X_train_num = create_model1(X_train, X_test, X_train_num,
categorical_headers_ints, cross_columns2)

    model1.compile(optimizer='adagrad',
                  loss='mean_squared_error',
                  metrics=['accuracy'])

    model1.fit(X_ints_train + [X_train_num], y_train, epochs=30, batch_size=50, verbose=1)
#this is just what i do with out cross validation
    yhat = np.round(model1.predict(X_ints_test + [X_test_num]))
    print(mt.confusion_matrix(y_test,yhat))
    acc_score1 = mt.accuracy_score(y_test, yhat)
    acc_scores1.append(acc_score1)#append all acc for k-fold
    fn_score1 = custom_score(y_test,yhat)
    fn_scores1.append(fn_score1)#append all fn for k-fold
    print("Accuracy: ", acc_score1)
    print("Fn : ", fn_score1)
print(fn_scores1)
print(acc_scores1)
```

```
Epoch 1/30
42388/42388 [==============================] - 7s 155us/step - loss: 0.1379 - acc: 
0.8054
Epoch 2/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1300 - acc: 
0.8156
Epoch 3/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1288 - acc: 
0.8173
Epoch 4/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1281 - acc: 
0.8187
Epoch 5/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1276 - acc: 
0.8196
Epoch 6/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1272 - acc: 
0.8206
Epoch 7/30
42388/42388 [==============================] - 4s 105us/step - loss: 0.1268 - acc: 
0.8209
Epoch 8/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1265 - acc: 
0.8201
Epoch 9/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1262 - acc: 
0.8214
Epoch 10/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1260 - acc: 
0.8224
Epoch 11/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1258 - acc: 
0.8225
Epoch 12/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1255 - acc: 
0.8223
Epoch 13/30
42388/42388 [==============================] - 4s 102us/step - loss: 0.1253 - acc: 
0.8235
Epoch 14/30
42388/42388 [==============================] - 4s 103us/step - loss: 0.1251 - acc: 
0.8234
Epoch 15/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1249 - acc: 
0.8229
Epoch 16/30
42388/42388 [==============================] - 4s 103us/step - loss: 0.1248 - acc: 
0.8241
Epoch 17/30
42388/42388 [==============================] - 4s 102us/step - loss: 0.1245 - acc: 
0.8239
Epoch 18/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1243 - acc: 
0.8248
Epoch 19/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1242 - acc: 
0.8240
Epoch 20/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1241 - acc: 
0.8246
Epoch 21/30
```

```
42388/42388 [==============================] - 5s 110us/step - loss: 0.1239 - acc:
0.8253
Epoch 22/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1237 - acc:
0.8257
Epoch 23/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1236 - acc:
0.8255
Epoch 24/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1235 - acc:
0.8251
Epoch 25/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1234 - acc:
0.8259
Epoch 26/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1232 - acc:
0.8264
Epoch 27/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1231 - acc:
0.8265
Epoch 28/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1230 - acc:
0.8262
Epoch 29/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1229 - acc:
0.8270
Epoch 30/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1228 - acc:
0.8272
[[6632  745]
 [1142 2079]]
Accuracy:  0.8219475372711832
Fn :  0.6051934287228405
Epoch 1/30
42388/42388 [==============================] - 7s 155us/step - loss: 0.1369 - acc:
0.8048
Epoch 2/30
42388/42388 [==============================] - 4s 103us/step - loss: 0.1303 - acc:
0.8155
Epoch 3/30
42388/42388 [==============================] - 4s 103us/step - loss: 0.1292 - acc:
0.8165
Epoch 4/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1285 - acc:
0.8174
Epoch 5/30
42388/42388 [==============================] - 5s 112us/step - loss: 0.1279 - acc:
0.8191
Epoch 6/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1275 - acc:
0.8193
Epoch 7/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1270 - acc:
0.8196
Epoch 8/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1267 - acc:
0.8204
Epoch 9/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1264 - acc:
0.8217
Epoch 10/30
```

```
42388/42388 [==============================] - 5s 108us/step - loss: 0.1260 - acc:
0.8222
Epoch 11/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1257 - acc:
0.8218
Epoch 12/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1254 - acc:
0.8231
Epoch 13/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1252 - acc:
0.8231
Epoch 14/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1249 - acc:
0.8237
Epoch 15/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1247 - acc:
0.8248
Epoch 16/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1245 - acc:
0.8246
Epoch 17/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1243 - acc:
0.8260
Epoch 18/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1241 - acc:
0.8253
Epoch 19/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1238 - acc:
0.8256
Epoch 20/30
42388/42388 [==============================] - 4s 100us/step - loss: 0.1238 - acc:
0.8259
Epoch 21/30
42388/42388 [==============================] - 4s 103us/step - loss: 0.1236 - acc:
0.8265
Epoch 22/30
42388/42388 [==============================] - 5s 122us/step - loss: 0.1234 - acc:
0.8268
Epoch 23/30
42388/42388 [==============================] - 4s 101us/step - loss: 0.1233 - acc:
0.8267
Epoch 24/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1231 - acc:
0.8275
Epoch 25/30
42388/42388 [==============================] - 4s 106us/step - loss: 0.1229 - acc:
0.8274
Epoch 26/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1227 - acc:
0.8272
Epoch 27/30
42388/42388 [==============================] - 5s 112us/step - loss: 0.1226 - acc:
0.8275
Epoch 28/30
42388/42388 [==============================] - 5s 112us/step - loss: 0.1225 - acc:
0.8277
Epoch 29/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1223 - acc:
0.8285
Epoch 30/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1222 - acc:
```

```
0.8278
[[6603  774]
 [1100 2121]]
Accuracy:  0.8231741838082657
Fn :  0.5869797225186766
Epoch 1/30
42388/42388 [==============================] - 7s 164us/step - loss: 0.1374 - acc:
0.8057
Epoch 2/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1300 - acc:
0.8159
Epoch 3/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1288 - acc:
0.8175
Epoch 4/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1281 - acc:
0.8173
Epoch 5/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1276 - acc:
0.8187
Epoch 6/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1270 - acc:
0.8204
Epoch 7/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1266 - acc:
0.8204
Epoch 8/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1262 - acc:
0.8213
Epoch 9/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1259 - acc:
0.8222
Epoch 10/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1257 - acc:
0.8225
Epoch 11/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1254 - acc:
0.8235
Epoch 12/30
42388/42388 [==============================] - 5s 113us/step - loss: 0.1251 - acc:
0.8233
Epoch 13/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1249 - acc:
0.8238
Epoch 14/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1247 - acc:
0.8245
Epoch 15/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1245 - acc:
0.8255
Epoch 16/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1242 - acc:
0.8254
Epoch 17/30
42388/42388 [==============================] - 5s 112us/step - loss: 0.1241 - acc:
0.8253
Epoch 18/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1239 - acc:
0.8261
Epoch 19/30
42388/42388 [==============================] - 5s 112us/step - loss: 0.1237 - acc:
```

```
0.8263
Epoch 20/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1235 - acc:
0.8276
Epoch 21/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1234 - acc:
0.8271
Epoch 22/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1232 - acc:
0.8281
Epoch 23/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1230 - acc:
0.8288
Epoch 24/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1228 - acc:
0.8283
Epoch 25/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1227 - acc:
0.8285
Epoch 26/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1225 - acc:
0.8295
Epoch 27/30
42388/42388 [==============================] - 5s 107us/step - loss: 0.1223 - acc:
0.8291
Epoch 28/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1222 - acc:
0.8295
Epoch 29/30
42388/42388 [==============================] - 5s 112us/step - loss: 0.1221 - acc:
0.8296
Epoch 30/30
42388/42388 [==============================] - 5s 116us/step - loss: 0.1219 - acc:
0.8302
[[6535  842]
 [1031 2190]]
Accuracy:  0.8232685412341951
Fn :  0.5504538174052322
Epoch 1/30
42388/42388 [==============================] - 7s 162us/step - loss: 0.1365 - acc:
0.8058
Epoch 2/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1302 - acc:
0.8166
Epoch 3/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1290 - acc:
0.8173
Epoch 4/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1282 - acc:
0.8190
Epoch 5/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1277 - acc:
0.8189
Epoch 6/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1272 - acc:
0.8200
Epoch 7/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1269 - acc:
0.8200
Epoch 8/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1265 - acc:
```

```
0.8204
Epoch 9/30
42388/42388 [==============================] - 5s 113us/step - loss: 0.1262 - acc:
0.8207
Epoch 10/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1259 - acc:
0.8220
Epoch 11/30
42388/42388 [==============================] - 5s 113us/step - loss: 0.1256 - acc:
0.8217
Epoch 12/30
42388/42388 [==============================] - 5s 116us/step - loss: 0.1254 - acc:
0.8228
Epoch 13/30
42388/42388 [==============================] - 5s 106us/step - loss: 0.1252 - acc:
0.8225
Epoch 14/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1249 - acc:
0.8234
Epoch 15/30
42388/42388 [==============================] - 5s 114us/step - loss: 0.1247 - acc:
0.8240
Epoch 16/30
42388/42388 [==============================] - 5s 113us/step - loss: 0.1246 - acc:
0.8239
Epoch 17/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1243 - acc:
0.8251
Epoch 18/30
42388/42388 [==============================] - 4s 103us/step - loss: 0.1241 - acc:
0.8247
Epoch 19/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1239 - acc:
0.8253
Epoch 20/30
42388/42388 [==============================] - 4s 104us/step - loss: 0.1238 - acc:
0.8256
Epoch 21/30
42388/42388 [==============================] - 4s 101us/step - loss: 0.1236 - acc:
0.8259
Epoch 22/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1234 - acc:
0.8256
Epoch 23/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1232 - acc:
0.8263
Epoch 24/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1230 - acc:
0.8268
Epoch 25/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1229 - acc:
0.8267
Epoch 26/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1227 - acc:
0.8272
Epoch 27/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1226 - acc:
0.8272
Epoch 28/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1224 - acc:
0.8283
```

```
Epoch 29/30
42388/42388 [==============================] - 5s 113us/step - loss: 0.1223 - acc:
0.8281
Epoch 30/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1221 - acc:
0.8284
[[6664  713]
 [1126 2095]]
Accuracy:  0.8264766937157955
Fn :  0.612289287656335
Epoch 1/30
42388/42388 [==============================] - 7s 170us/step - loss: 0.1367 - acc:
0.8064
Epoch 2/30
42388/42388 [==============================] - 5s 113us/step - loss: 0.1297 - acc:
0.8181
Epoch 3/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1285 - acc:
0.8185
Epoch 4/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1278 - acc:
0.8195
Epoch 5/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1272 - acc:
0.8208
Epoch 6/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1267 - acc:
0.8216
Epoch 7/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1263 - acc:
0.8228
Epoch 8/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1259 - acc:
0.8229
Epoch 9/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1256 - acc:
0.8235
Epoch 10/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1253 - acc:
0.8235
Epoch 11/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1250 - acc:
0.8245
Epoch 12/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1247 - acc:
0.8246
Epoch 13/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1245 - acc:
0.8249
Epoch 14/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1242 - acc:
0.8258
Epoch 15/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1240 - acc:
0.8260
Epoch 16/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1238 - acc:
0.8263
Epoch 17/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1235 - acc:
0.8272
```

```
Epoch 18/30
42388/42388 [==============================] - 5s 111us/step - loss: 0.1234 - acc:
0.8270
Epoch 19/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1231 - acc:
0.8272
Epoch 20/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1230 - acc:
0.8276
Epoch 21/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1228 - acc:
0.8278
Epoch 22/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1226 - acc:
0.8286
Epoch 23/30
42388/42388 [==============================] - 5s 110us/step - loss: 0.1225 - acc:
0.8291
Epoch 24/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1224 - acc:
0.8291
Epoch 25/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1222 - acc:
0.8293
Epoch 26/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1221 - acc:
0.8292
Epoch 27/30
42388/42388 [==============================] - 5s 108us/step - loss: 0.1219 - acc:
0.8299
Epoch 28/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1218 - acc:
0.8288
Epoch 29/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1215 - acc:
0.8304
Epoch 30/30
42388/42388 [==============================] - 5s 109us/step - loss: 0.1215 - acc:
0.8306
[[6640  737]
 [1137 2084]]
Accuracy:  0.8231741838082657
Fn :  0.6067235859124867
```

Different layers

Model1 with 7 layer in Deep and cross combo1

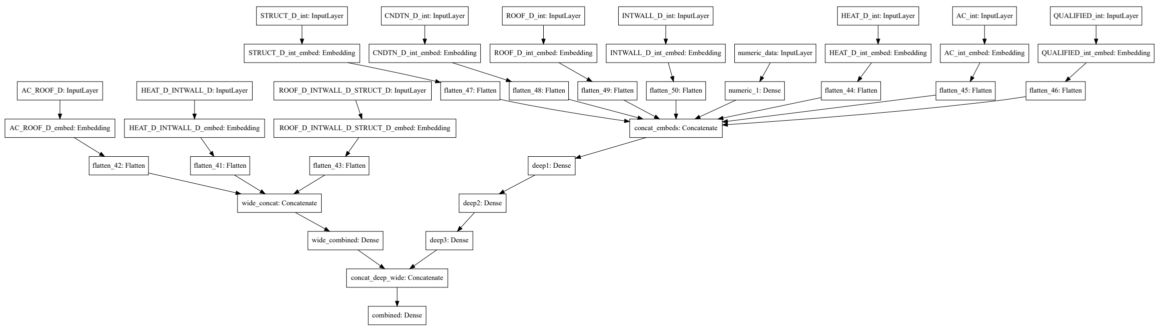
In [447]:

```
#model1
from keras.layers import concatenate
from sklearn.preprocessing import OneHotEncoder

#'HEAT_D','AC','QUALIFIED','STRUCT_D','CNDTN_D','ROOF_D','INTWALL_D'
#"BATHRM", "BEDRM", "AYB_year", "YR_RMDL_year", "EYB_year", "STORAGE", "GBA", "KITCHENS"
cross_columns1 = [['HEAT_D','AC'],
                 ['QUALIFIED', 'CNDTN_D'],
                 ['ROOF_D','INTWALL_D']]

# and save off the numeric features
X_train_num =  df_train[numeric_headers].values
X_test_num = df_test[numeric_headers].values


def create_model2(X_train, X_test, X_train_num, categorical_headers_ints, cross_columns=cross_co
lumns1):
    embed_branches = []
    X_ints_train = []
    X_ints_test = []
    all_inputs = []
    all_wide_branch_outputs = []

    for cols in cross_columns:
    # encode crossed columns as ints for the embedding
        enc = LabelEncoder()

    # create crossed labels
        X_crossed_train = df_train[cols].apply(lambda x: '_'.join(x), axis=1)
        X_crossed_test = df_test[cols].apply(lambda x: '_'.join(x), axis=1)

        enc.fit(np.hstack((X_crossed_train.values,  X_crossed_test.values)))
        X_crossed_train = enc.transform(X_crossed_train)
        X_crossed_test = enc.transform(X_crossed_test)
        X_ints_train.append( X_crossed_train )
        X_ints_test.append( X_crossed_test )

    # get the number of categories
        N = max(X_ints_train[-1]+1) # same as the max(df_train[col])

    # create embedding branch from the number of categories
        inputs = Input(shape=(1,),dtype='int32', name = '_'.join(cols))
        all_inputs.append(inputs)
        x = Embedding(input_dim=N,
                  output_dim=int(np.sqrt(N)),
                  input_length=1, name = '_'.join(cols)+'_embed')(inputs)
        x = Flatten()(x)
        all_wide_branch_outputs.append(x)

# merge the branches together
    wide_branch = concatenate(all_wide_branch_outputs, name='wide_concat')
    wide_branch = Dense(units=1,activation='sigmoid',name='wide_combined')(wide_branch)

# reset this input branch
    all_deep_branch_outputs = []
# add in the embeddings
    for col in categorical_headers_ints:
    # encode as ints for the embedding
        X_ints_train.append( df_train[col].values )
        X_ints_test.append( df_test[col].values )

    # get the number of categories
        N = max(X_ints_train[-1]+1) # same as the max(df_train[col])

    # create embedding branch from the number of categories
        inputs = Input(shape=(1,),dtype='int32', name=col)
        all_inputs.append(inputs)
        x = Embedding(input_dim=N,
                  output_dim=int(np.sqrt(N)),
                  input_length=1, name=col+'_embed')(inputs)
        x = Flatten()(x)
        all_deep_branch_outputs.append(x)

# also get a dense branch of the numeric features
    all_inputs.append(Input(shape=(X_train_num.shape[1],),
                            sparse=False,
                            name='numeric_data'))

    x = Dense(units=20, activation='relu',name='numeric_1')(all_inputs[-1])
    all_deep_branch_outputs.append( x )

# merge the deep branches together
    deep_branch = concatenate(all_deep_branch_outputs,name='concat_embeds')
    deep_branch = Dense(units=50,activation='relu', name='deep1')(deep_branch)
    deep_branch = Dense(units=25,activation='relu', name='deep2')(deep_branch)
    deep_branch = Dense(units=10,activation='relu', name='deep3')(deep_branch)
    deep_branch = Dense(units=10,activation='relu', name='deep4')(deep_branch)
    deep_branch = Dense(units=10,activation='relu', name='deep5')(deep_branch)
    deep_branch = Dense(units=10,activation='relu', name='deep6')(deep_branch)
    deep_branch = Dense(units=10,activation='relu', name='deep7')(deep_branch)

    final_branch = concatenate([wide_branch, deep_branch],name='concat_deep_wide')
    final_branch = Dense(units=1,activation='sigmoid',name='combined')(final_branch)

    model2 = Model(inputs=all_inputs, outputs=final_branch)
    return model2, X_ints_train, X_ints_test, X_train_num
```

In [390]:

```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

# you will need to install pydot properly on your machine to get this running
SVG(model_to_dot(model).create(prog='dot', format='svg'))
```

Out[390]:



In [391]:

```
model2.compile(optimizer='adagrad',
              loss='mean_squared_error',
              metrics=['accuracy'])
```

In [392]:

```
# lets also add the history variable to see how we are doing
# and lets add a validation set to keep track of our progress
history2 = model2.fit(X_ints_train+ [X_train_num],
                    y_train,
                    epochs=30,
                    batch_size=50,
                    verbose=1,
                    validation_data = (X_ints_test + [X_test_num], y_test))
```

```
Train on 42388 samples, validate on 10598 samples
Epoch 1/30
42388/42388 [==============================] - 6s 139us/step - loss: 0.1373 - acc:
0.8046 - val_loss: 0.1313 - val_acc: 0.8119
Epoch 2/30
42388/42388 [==============================] - 5s 119us/step - loss: 0.1301 - acc:
0.8168 - val_loss: 0.1302 - val_acc: 0.8135
Epoch 3/30
42388/42388 [==============================] - 5s 119us/step - loss: 0.1290 - acc:
0.8181 - val_loss: 0.1279 - val_acc: 0.8178
Epoch 4/30
42388/42388 [==============================] - 5s 119us/step - loss: 0.1282 - acc:
0.8192 - val_loss: 0.1278 - val_acc: 0.8168
Epoch 5/30
42388/42388 [==============================] - 5s 118us/step - loss: 0.1276 - acc:
0.8195 - val_loss: 0.1274 - val_acc: 0.8168
Epoch 6/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1272 - acc:
0.8195 - val_loss: 0.1269 - val_acc: 0.8199
Epoch 7/30
42388/42388 [==============================] - 5s 116us/step - loss: 0.1266 - acc:
0.8205 - val_loss: 0.1281 - val_acc: 0.8165
Epoch 8/30
42388/42388 [==============================] - 5s 117us/step - loss: 0.1263 - acc:
0.8214 - val_loss: 0.1266 - val_acc: 0.8192
Epoch 9/30
42388/42388 [==============================] - 5s 119us/step - loss: 0.1259 - acc:
0.8221 - val_loss: 0.1264 - val_acc: 0.8199
Epoch 10/30
42388/42388 [==============================] - 5s 120us/step - loss: 0.1255 - acc:
0.8223 - val_loss: 0.1261 - val_acc: 0.8209
Epoch 11/30
42388/42388 [==============================] - 5s 120us/step - loss: 0.1253 - acc:
0.8223 - val_loss: 0.1254 - val_acc: 0.8210
Epoch 12/30
42388/42388 [==============================] - 5s 117us/step - loss: 0.1250 - acc:
0.8228 - val_loss: 0.1254 - val_acc: 0.8214
Epoch 13/30
42388/42388 [==============================] - 5s 121us/step - loss: 0.1246 - acc:
0.8244 - val_loss: 0.1259 - val_acc: 0.8209
Epoch 14/30
42388/42388 [==============================] - 5s 120us/step - loss: 0.1244 - acc:
0.8243 - val_loss: 0.1252 - val_acc: 0.8219
Epoch 15/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1241 - acc:
0.8246 - val_loss: 0.1250 - val_acc: 0.8228
Epoch 16/30
42388/42388 [==============================] - 5s 123us/step - loss: 0.1239 - acc:
0.8260 - val_loss: 0.1249 - val_acc: 0.8212
Epoch 17/30
42388/42388 [==============================] - 5s 114us/step - loss: 0.1236 - acc:
0.8255 - val_loss: 0.1253 - val_acc: 0.8221
Epoch 18/30
42388/42388 [==============================] - 5s 115us/step - loss: 0.1234 - acc:
0.8264 - val_loss: 0.1246 - val_acc: 0.8225
Epoch 19/30
42388/42388 [==============================] - 5s 114us/step - loss: 0.1231 - acc:
0.8269 - val_loss: 0.1253 - val_acc: 0.8218
Epoch 20/30
42388/42388 [==============================] - 5s 114us/step - loss: 0.1229 - acc:
0.8275 - val_loss: 0.1245 - val_acc: 0.8220
```

```
Epoch 21/30
42388/42388 [==============================] - 5s 114us/step - loss: 0.1226 - acc:
0.8281 - val_loss: 0.1246 - val_acc: 0.8222
Epoch 22/30
42388/42388 [==============================] - 5s 114us/step - loss: 0.1224 - acc:
0.8283 - val_loss: 0.1245 - val_acc: 0.8225
Epoch 23/30
42388/42388 [==============================] - 5s 114us/step - loss: 0.1222 - acc:
0.8284 - val_loss: 0.1243 - val_acc: 0.8240
Epoch 24/30
42388/42388 [==============================] - 5s 114us/step - loss: 0.1219 - acc:
0.8288 - val_loss: 0.1243 - val_acc: 0.8236
Epoch 25/30
42388/42388 [==============================] - 5s 114us/step - loss: 0.1217 - acc:
0.8292 - val_loss: 0.1244 - val_acc: 0.8224
Epoch 26/30
42388/42388 [==============================] - 5s 114us/step - loss: 0.1214 - acc:
0.8295 - val_loss: 0.1244 - val_acc: 0.8238
Epoch 27/30
42388/42388 [==============================] - 5s 114us/step - loss: 0.1212 - acc:
0.8308 - val_loss: 0.1239 - val_acc: 0.8246
Epoch 28/30
42388/42388 [==============================] - 5s 114us/step - loss: 0.1210 - acc:
0.8308 - val_loss: 0.1240 - val_acc: 0.8226
Epoch 29/30
42388/42388 [==============================] - 5s 114us/step - loss: 0.1207 - acc:
0.8307 - val_loss: 0.1246 - val_acc: 0.8235
Epoch 30/30
42388/42388 [==============================] - 5s 114us/step - loss: 0.1204 - acc:
0.8323 - val_loss: 0.1236 - val_acc: 0.8259
```

In [393]:

```
#accuracy score
yhat = np.round(model2.predict(X_ints_test + [X_test_num]))
print(mt.confusion_matrix(y_test,yhat), 'Accuracy ',mt.accuracy_score(y_test,yhat))
#my FN score
print('FN score ',custom_score(y_test,yhat))
```

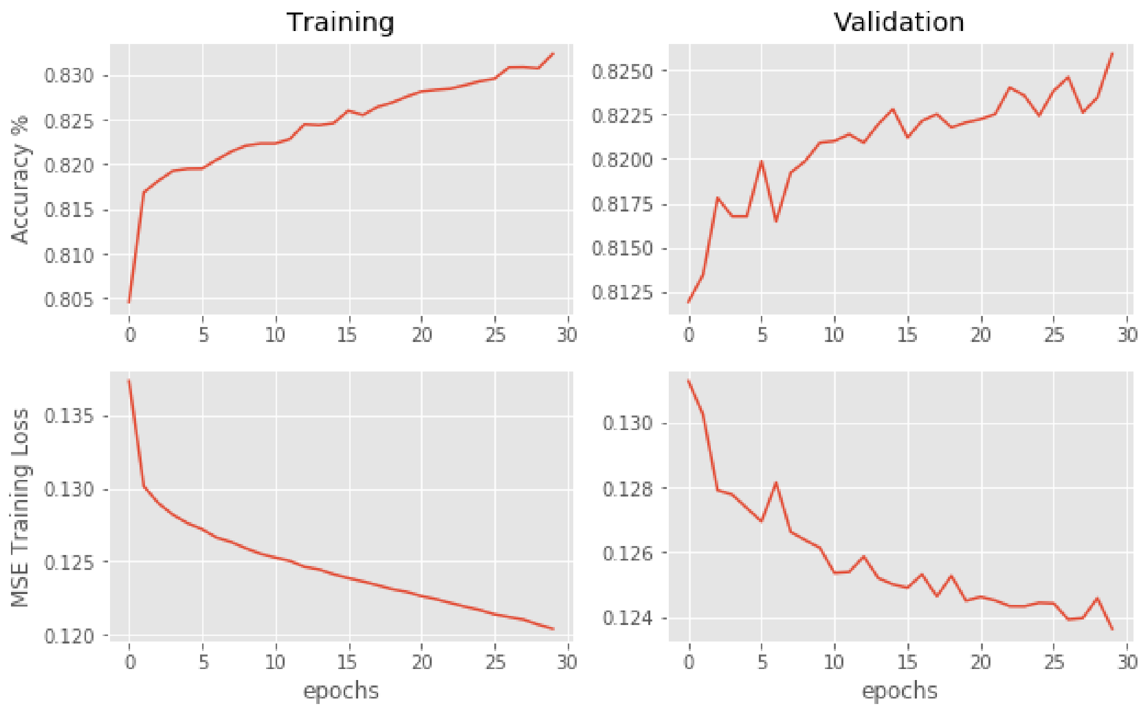
```
[[6627  750]
 [1095 2126]] Accuracy  0.8259105491602189
FN score  0.5934959349593496
```

In [394]:

```
from matplotlib import pyplot as plt

%matplotlib inline

plt.figure(figsize=(10,6))
plt.subplot(2,2,1)
plt.plot(history2.history['acc'])

plt.ylabel('Accuracy %')
plt.title('Training')
plt.subplot(2,2,2)
plt.plot(history2.history['val_acc'])
plt.title('Validation')

plt.subplot(2,2,3)
plt.plot(history2.history['loss'])
plt.ylabel('MSE Training Loss')
plt.xlabel('epochs')

plt.subplot(2,2,4)
plt.plot(history2.history['val_loss'])
plt.xlabel('epochs')
```

Out[394]:

```
Text(0.5, 0, 'epochs')
```



For train vs loss, I think it converge, the line reach the bottom. For validation, the line boucing around but it converge. In general, they converge. I think 30 epochs is fine.

In [448]:

```
#https://github.com/Thakugan/machine-learning-notebooks/blob/master/6-wide-and-deep-networks/mus
hroom-hunting.ipynb
num_folds = 5

acc_scores2 =[]
fn_scores2 = []

skf = StratifiedKFold(n_splits=num_folds, shuffle=True)
for i, (train, test) in enumerate(skf.split(X, y)):#here I used corss validation on my model

    model2, X_ints_train, X_ints_test, X_train_num = create_model2(X_train, X_test, X_train_num,
categorical_headers_ints, cross_columns1)

    model2.compile(optimizer='adagrad',
                  loss='mean_squared_error',
                  metrics=['accuracy'])

    model2.fit(X_ints_train + [X_train_num], y_train, epochs=30, batch_size=50, verbose=1)
#this is just what i do with out cross validation
    yhat = np.round(model2.predict(X_ints_test + [X_test_num]))
    print(mt.confusion_matrix(y_test,yhat))
    acc_score2 = mt.accuracy_score(y_test, yhat)
    acc_scores2.append(acc_score2)#append all acc for k-fold
    fn_score2 = custom_score(y_test,yhat)
    fn_scores2.append(fn_score2)#append all fn for k-fold
    print("Accuracy: ", acc_score2)
    print("Fn : ", fn_score2)
print(fn_scores2)
print(acc_scores2)
```

```
Epoch 1/30
42388/42388 [==============================] - 8s 185us/step - loss: 0.1395 - acc:
0.8014
Epoch 2/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1298 - acc:
0.8158
Epoch 3/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1286 - acc:
0.8168
Epoch 4/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1279 - acc:
0.8185
Epoch 5/30
42388/42388 [==============================] - 5s 130us/step - loss: 0.1273 - acc:
0.8189
Epoch 6/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1268 - acc:
0.8204
Epoch 7/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1265 - acc:
0.8211
Epoch 8/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1261 - acc:
0.8214
Epoch 9/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1258 - acc:
0.8220
Epoch 10/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1255 - acc:
0.8227
Epoch 11/30
42388/42388 [==============================] - 10s 227us/step - loss: 0.1252 - ac
c: 0.8227
Epoch 12/30
42388/42388 [==============================] - 5s 124us/step - loss: 0.1249 - acc:
0.8225
Epoch 13/30
42388/42388 [==============================] - 5s 124us/step - loss: 0.1247 - acc:
0.8233
Epoch 14/30
42388/42388 [==============================] - 5s 123us/step - loss: 0.1244 - acc:
0.8234
Epoch 15/30
42388/42388 [==============================] - 5s 123us/step - loss: 0.1242 - acc:
0.8239
Epoch 16/30
42388/42388 [==============================] - 5s 124us/step - loss: 0.1239 - acc:
0.8241
Epoch 17/30
42388/42388 [==============================] - 5s 124us/step - loss: 0.1238 - acc:
0.8247
Epoch 18/30
42388/42388 [==============================] - 5s 123us/step - loss: 0.1235 - acc:
0.8244
Epoch 19/30
42388/42388 [==============================] - 5s 124us/step - loss: 0.1234 - acc:
0.8252
Epoch 20/30
42388/42388 [==============================] - 5s 123us/step - loss: 0.1232 - acc:
0.8251
Epoch 21/30
```

```
42388/42388 [==============================] - 5s 125us/step - loss: 0.1229 - acc:
0.8258
Epoch 22/30
42388/42388 [==============================] - 5s 124us/step - loss: 0.1228 - acc:
0.8264
Epoch 23/30
42388/42388 [==============================] - 5s 124us/step - loss: 0.1226 - acc:
0.8261
Epoch 24/30
42388/42388 [==============================] - 5s 124us/step - loss: 0.1224 - acc:
0.8270
Epoch 25/30
42388/42388 [==============================] - 5s 124us/step - loss: 0.1222 - acc:
0.8270
Epoch 26/30
42388/42388 [==============================] - 5s 124us/step - loss: 0.1220 - acc:
0.8274
Epoch 27/30
42388/42388 [==============================] - 5s 124us/step - loss: 0.1218 - acc:
0.8280
Epoch 28/30
42388/42388 [==============================] - 5s 124us/step - loss: 0.1216 - acc:
0.8281
Epoch 29/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1215 - acc:
0.8282
Epoch 30/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1212 - acc:
0.8291
[[6619  758]
 [1101 2120]]
Accuracy:  0.8245895451972071
Fn :  0.5922538999462076
Epoch 1/30
42388/42388 [==============================] - 8s 184us/step - loss: 0.1384 - acc:
0.8027
Epoch 2/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1303 - acc:
0.8168
Epoch 3/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1287 - acc:
0.8194
Epoch 4/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1280 - acc:
0.8188
Epoch 5/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1275 - acc:
0.8207
Epoch 6/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1270 - acc:
0.8216
Epoch 7/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1265 - acc:
0.8212
Epoch 8/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1262 - acc:
0.8221
Epoch 9/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1258 - acc:
0.8219
Epoch 10/30
```

```
42388/42388 [==============================] - 5s 125us/step - loss: 0.1255 - acc:
0.8226
Epoch 11/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1252 - acc:
0.8233
Epoch 12/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1248 - acc:
0.8238
Epoch 13/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1245 - acc:
0.8238
Epoch 14/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1243 - acc:
0.8246
Epoch 15/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1240 - acc:
0.8248
Epoch 16/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1237 - acc:
0.8264
Epoch 17/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1234 - acc:
0.8260
Epoch 18/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1231 - acc:
0.8262
Epoch 19/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1230 - acc:
0.8263
Epoch 20/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1227 - acc:
0.8269
Epoch 21/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1225 - acc:
0.8271
Epoch 22/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1222 - acc:
0.8269
Epoch 23/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1220 - acc:
0.8281
Epoch 24/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1218 - acc:
0.8280
Epoch 25/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1215 - acc:
0.8300
Epoch 26/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1213 - acc:
0.8292
Epoch 27/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1211 - acc:
0.8293
Epoch 28/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1209 - acc:
0.8309
Epoch 29/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1207 - acc:
0.8307
Epoch 30/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1205 - acc:
```

```
0.8301
[[6706  671]
 [1200 2021]]
Accuracy:  0.823457256086054
Fn :  0.6413682522715125
Epoch 1/30
42388/42388 [==============================] - 8s 189us/step - loss: 0.1384 - acc:
0.8030
Epoch 2/30
42388/42388 [==============================] - 5s 130us/step - loss: 0.1310 - acc:
0.8153
Epoch 3/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1298 - acc:
0.8175
Epoch 4/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1292 - acc:
0.8188
Epoch 5/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1287 - acc:
0.8183
Epoch 6/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1281 - acc:
0.8193
Epoch 7/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1278 - acc:
0.8197
Epoch 8/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1275 - acc:
0.8201
Epoch 9/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1271 - acc:
0.8207
Epoch 10/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1268 - acc:
0.8215
Epoch 11/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1265 - acc:
0.8215
Epoch 12/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1262 - acc:
0.8214
Epoch 13/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1260 - acc:
0.8220
Epoch 14/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1257 - acc:
0.8224
Epoch 15/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1255 - acc:
0.8226
Epoch 16/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1253 - acc:
0.8230
Epoch 17/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1250 - acc:
0.8233
Epoch 18/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1248 - acc:
0.8239
Epoch 19/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1246 - acc:
```

```
0.8247
Epoch 20/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1244 - acc:
0.8243
Epoch 21/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1243 - acc:
0.8239
Epoch 22/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1240 - acc:
0.8248
Epoch 23/30
42388/42388 [==============================] - 5s 124us/step - loss: 0.1237 - acc:
0.8250
Epoch 24/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1236 - acc:
0.8252
Epoch 25/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1235 - acc:
0.8254
Epoch 26/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1233 - acc:
0.8262
Epoch 27/30
42388/42388 [==============================] - ETA: 0s - loss: 0.1229 - acc: 0.826
- 5s 126us/step - loss: 0.1230 - acc: 0.8262
Epoch 28/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1230 - acc:
0.8264
Epoch 29/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1228 - acc:
0.8267
Epoch 30/30
42388/42388 [==============================] - 5s 125us/step - loss: 0.1227 - acc:
0.8273
[[6653  724]
 [1167 2054]]
Accuracy:  0.8215701075674655
Fn :  0.6171337916446324
Epoch 1/30
42388/42388 [==============================] - 8s 188us/step - loss: 0.1403 - acc:
0.8012
Epoch 2/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1313 - acc:
0.8143
Epoch 3/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1301 - acc:
0.8166
Epoch 4/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1292 - acc:
0.8170
Epoch 5/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1284 - acc:
0.8174
Epoch 6/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1279 - acc:
0.8181
Epoch 7/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1275 - acc:
0.8195
Epoch 8/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1271 - acc:
```

```
0.8195
Epoch 9/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1266 - acc:
0.8194
Epoch 10/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1263 - acc:
0.8211
Epoch 11/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1260 - acc:
0.8214
Epoch 12/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1259 - acc:
0.8203
Epoch 13/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1257 - acc:
0.8208
Epoch 14/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1254 - acc:
0.8221
Epoch 15/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1252 - acc:
0.8227
Epoch 16/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1250 - acc:
0.8227
Epoch 17/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1248 - acc:
0.8227
Epoch 18/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1246 - acc:
0.8226
Epoch 19/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1244 - acc:
0.8233
Epoch 20/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1242 - acc:
0.8232
Epoch 21/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1241 - acc:
0.8235
Epoch 22/30
42388/42388 [==============================] - 5s 126us/step - loss: 0.1240 - acc:
0.8247
Epoch 23/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1238 - acc:
0.8245
Epoch 24/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1236 - acc:
0.8246
Epoch 25/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1235 - acc:
0.8248
Epoch 26/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1233 - acc:
0.8247
Epoch 27/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1232 - acc:
0.8257
Epoch 28/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1231 - acc:
0.8260
```

```
Epoch 29/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1229 - acc:
0.8262
Epoch 30/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1228 - acc:
0.8253
[[6642  735]
 [1170 2051]]
Accuracy:  0.8202491036044537
Fn :  0.6141732283464567
Epoch 1/30
42388/42388 [==============================] - 8s 193us/step - loss: 0.1382 - acc:
0.8016
Epoch 2/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1305 - acc:
0.8145
Epoch 3/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1295 - acc:
0.8160
Epoch 4/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1287 - acc:
0.8177
Epoch 5/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1281 - acc:
0.8185
Epoch 6/30
42388/42388 [==============================] - 5s 130us/step - loss: 0.1277 - acc:
0.8184
Epoch 7/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1272 - acc:
0.8193
Epoch 8/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1269 - acc:
0.8195
Epoch 9/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1266 - acc:
0.8196
Epoch 10/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1262 - acc:
0.8202
Epoch 11/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1260 - acc:
0.8209
Epoch 12/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1256 - acc:
0.8216
Epoch 13/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1254 - acc:
0.8214
Epoch 14/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1251 - acc:
0.8224
Epoch 15/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1248 - acc:
0.8223
Epoch 16/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1245 - acc:
0.8223
Epoch 17/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1244 - acc:
0.8228
```

```
Epoch 18/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1242 - acc:
0.8233
Epoch 19/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1239 - acc:
0.8233
Epoch 20/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1238 - acc:
0.8232
Epoch 21/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1235 - acc:
0.8243
Epoch 22/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1233 - acc:
0.8248
Epoch 23/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1232 - acc:
0.8251
Epoch 24/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1229 - acc:
0.8259
Epoch 25/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1228 - acc:
0.8264
Epoch 26/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1227 - acc:
0.8263
Epoch 27/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1225 - acc:
0.8270
Epoch 28/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1223 - acc:
0.8270
Epoch 29/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1222 - acc:
0.8262
Epoch 30/30
42388/42388 [==============================] - 6s 133us/step - loss: 0.1220 - acc:
0.8271
[[6649  728]
 [1163 2058]]
Accuracy:  0.8215701075674655
Fn :  0.6150185087255421
[0.5922538999462076, 0.6413682522715125, 0.6171337916446324, 0.6141732283464567,
0.6150185087255421]
[0.8245895451972071, 0.823457256086054, 0.8215701075674655, 0.8202491036044537, 0.
8215701075674655]
```

Model2 with 7 layer in Deep and cross combo2

In [449]:

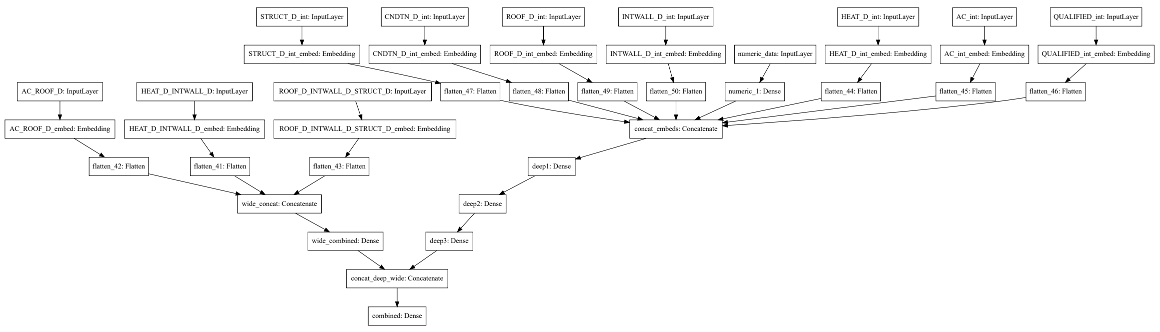
```
#model2
from keras.layers import concatenate
from sklearn.preprocessing import OneHotEncoder

#'HEAT_D','AC','QUALIFIED','STRUCT_D','CNDTN_D','ROOF_D','INTWALL_D'
#"BATHRM","BEDRM","AYB_year","YR_RMDL_year","EYB_year","STORAGE","GBA","KITCHENS"
cross_columns2 = [['HEAT_D','INTWALL_D'],
                  ['AC', 'ROOF_D'],
                  ['ROOF_D','INTWALL_D','STRUCT_D']]

# and save off the numeric features
X_train_num =  df_train[numeric_headers].values
X_test_num = df_test[numeric_headers].values

#def create_model this just add in frount of original Dr. Larson code, doing this for cross vali
dation
def create_model3(X_train, X_test, X_train_num, categorical_headers_ints, cross_columns=cross_co
lumns2):
    embed_branches = []
    X_ints_train = []
    X_ints_test = []
    all_inputs = []
    all_wide_branch_outputs = []

    for cols in cross_columns:
    # encode crossed columns as ints for the embedding
        enc = LabelEncoder()

    # create crossed labels
        X_crossed_train = df_train[cols].apply(lambda x: '_'.join(x), axis=1)
        X_crossed_test = df_test[cols].apply(lambda x: '_'.join(x), axis=1)

        enc.fit(np.hstack((X_crossed_train.values,  X_crossed_test.values)))
        X_crossed_train = enc.transform(X_crossed_train)
        X_crossed_test = enc.transform(X_crossed_test)
        X_ints_train.append( X_crossed_train )
        X_ints_test.append( X_crossed_test )

    # get the number of categories
        N = max(X_ints_train[-1]+1) # same as the max(df_train[col])

    # create embedding branch from the number of categories
        inputs = Input(shape=(1,),dtype='int32', name = '_'.join(cols))
        all_inputs.append(inputs)
        x = Embedding(input_dim=N,
                      output_dim=int(np.sqrt(N)),
                      input_length=1, name = '_'.join(cols)+'_embed')(inputs)
        x = Flatten()(x)
        all_wide_branch_outputs.append(x)

# merge the branches together
    wide_branch = concatenate(all_wide_branch_outputs, name='wide_concat')
    wide_branch = Dense(units=1,activation='sigmoid',name='wide_combined')(wide_branch)

# reset this input branch
    all_deep_branch_outputs = []
# add in the embeddings
    for col in categorical_headers_ints:
    # encode as ints for the embedding
```

```
        X_ints_train.append( df_train[col].values )
        X_ints_test.append( df_test[col].values )

    # get the number of categories
        N = max(X_ints_train[-1]+1) # same as the max(df_train[col])

    # create embedding branch from the number of categories
        inputs = Input(shape=(1,),dtype='int32', name=col)
        all_inputs.append(inputs)
        x = Embedding(input_dim=N,
                      output_dim=int(np.sqrt(N)),
                      input_length=1, name=col+'_embed')(inputs)
        x = Flatten()(x)
        all_deep_branch_outputs.append(x)

# also get a dense branch of the numeric features
    all_inputs.append(Input(shape=(X_train_num.shape[1],),
                            sparse=False,
                            name='numeric_data'))

    x = Dense(units=20, activation='relu',name='numeric_1')(all_inputs[-1])
    all_deep_branch_outputs.append( x )

# merge the deep branches together
    deep_branch = concatenate(all_deep_branch_outputs,name='concat_embeds')
    deep_branch = Dense(units=50,activation='relu', name='deep1')(deep_branch)
    deep_branch = Dense(units=25,activation='relu', name='deep2')(deep_branch)
    deep_branch = Dense(units=10,activation='relu', name='deep3')(deep_branch)
    deep_branch = Dense(units=10,activation='relu', name='deep4')(deep_branch)
    deep_branch = Dense(units=10,activation='relu', name='deep5')(deep_branch)
    deep_branch = Dense(units=10,activation='relu', name='deep6')(deep_branch)
    deep_branch = Dense(units=10,activation='relu', name='deep7')(deep_branch)

    final_branch = concatenate([wide_branch, deep_branch],name='concat_deep_wide')
    final_branch = Dense(units=1,activation='sigmoid',name='combined')(final_branch)

    model2 = Model(inputs=all_inputs, outputs=final_branch)
    return model2, X_ints_train, X_ints_test, X_train_num
```

In [396]:

```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

# you will need to install pydot properly on your machine to get this running
SVG(model_to_dot(model).create(prog='dot', format='svg'))
```

Out[396]:



In [397]:

```
model3.compile(optimizer='adagrad',
              loss='mean_squared_error',
              metrics=['accuracy'])
```

In [398]:

```
# lets also add the history variable to see how we are doing
# and lets add a validation set to keep track of our progress
history3 = model3.fit(X_ints_train+ [X_train_num],
                    y_train,
                    epochs=30,
                    batch_size=50,
                    verbose=1,
                    validation_data = (X_ints_test + [X_test_num], y_test))
```

```
Train on 42388 samples, validate on 10598 samples
Epoch 1/30
42388/42388 [==============================] - 6s 144us/step - loss: 0.1410 - acc:
0.7987 - val_loss: 0.1302 - val_acc: 0.8169
Epoch 2/30
42388/42388 [==============================] - 5s 123us/step - loss: 0.1308 - acc:
0.8161 - val_loss: 0.1289 - val_acc: 0.8180
Epoch 3/30
42388/42388 [==============================] - 5s 121us/step - loss: 0.1295 - acc:
0.8182 - val_loss: 0.1289 - val_acc: 0.8183
Epoch 4/30
42388/42388 [==============================] - 5s 122us/step - loss: 0.1288 - acc:
0.8181 - val_loss: 0.1281 - val_acc: 0.8173
Epoch 5/30
42388/42388 [==============================] - 5s 120us/step - loss: 0.1283 - acc:
0.8176 - val_loss: 0.1278 - val_acc: 0.8180
Epoch 6/30
42388/42388 [==============================] - 5s 123us/step - loss: 0.1279 - acc:
0.8194 - val_loss: 0.1276 - val_acc: 0.8176
Epoch 7/30
42388/42388 [==============================] - 5s 123us/step - loss: 0.1275 - acc:
0.8194 - val_loss: 0.1272 - val_acc: 0.8189
Epoch 8/30
42388/42388 [==============================] - 5s 122us/step - loss: 0.1272 - acc:
0.8191 - val_loss: 0.1273 - val_acc: 0.8186
Epoch 9/30
42388/42388 [==============================] - 5s 122us/step - loss: 0.1269 - acc:
0.8197 - val_loss: 0.1269 - val_acc: 0.8191
Epoch 10/30
42388/42388 [==============================] - 5s 121us/step - loss: 0.1266 - acc:
0.8211 - val_loss: 0.1267 - val_acc: 0.8209
Epoch 11/30
42388/42388 [==============================] - 5s 119us/step - loss: 0.1263 - acc:
0.8212 - val_loss: 0.1269 - val_acc: 0.8193
Epoch 12/30
42388/42388 [==============================] - 5s 120us/step - loss: 0.1261 - acc:
0.8217 - val_loss: 0.1266 - val_acc: 0.8205
Epoch 13/30
42388/42388 [==============================] - 5s 120us/step - loss: 0.1259 - acc:
0.8216 - val_loss: 0.1263 - val_acc: 0.8197
Epoch 14/30
42388/42388 [==============================] - 5s 119us/step - loss: 0.1257 - acc:
0.8221 - val_loss: 0.1264 - val_acc: 0.8208
Epoch 15/30
42388/42388 [==============================] - 5s 120us/step - loss: 0.1255 - acc:
0.8219 - val_loss: 0.1260 - val_acc: 0.8215
Epoch 16/30
42388/42388 [==============================] - 5s 120us/step - loss: 0.1253 - acc:
0.8235 - val_loss: 0.1261 - val_acc: 0.8225
Epoch 17/30
42388/42388 [==============================] - 5s 114us/step - loss: 0.1251 - acc:
0.8225 - val_loss: 0.1259 - val_acc: 0.8216
Epoch 18/30
42388/42388 [==============================] - 5s 115us/step - loss: 0.1250 - acc:
0.8232 - val_loss: 0.1258 - val_acc: 0.8222
Epoch 19/30
42388/42388 [==============================] - 5s 115us/step - loss: 0.1248 - acc:
0.8244 - val_loss: 0.1257 - val_acc: 0.8227
Epoch 20/30
42388/42388 [==============================] - 5s 115us/step - loss: 0.1246 - acc:
0.8239 - val_loss: 0.1255 - val_acc: 0.8229
```

```
Epoch 21/30
42388/42388 [==============================] - 5s 116us/step - loss: 0.1245 - acc:
0.8240 - val_loss: 0.1255 - val_acc: 0.8224
Epoch 22/30
42388/42388 [==============================] - 5s 114us/step - loss: 0.1243 - acc:
0.8253 - val_loss: 0.1253 - val_acc: 0.8220
Epoch 23/30
42388/42388 [==============================] - 5s 117us/step - loss: 0.1241 - acc:
0.8252 - val_loss: 0.1252 - val_acc: 0.8223
Epoch 24/30
42388/42388 [==============================] - 5s 115us/step - loss: 0.1240 - acc:
0.8258 - val_loss: 0.1252 - val_acc: 0.8225
Epoch 25/30
42388/42388 [==============================] - 5s 116us/step - loss: 0.1239 - acc:
0.8255 - val_loss: 0.1251 - val_acc: 0.8216
Epoch 26/30
42388/42388 [==============================] - 5s 115us/step - loss: 0.1238 - acc:
0.8263 - val_loss: 0.1252 - val_acc: 0.8219
Epoch 27/30
42388/42388 [==============================] - 5s 116us/step - loss: 0.1236 - acc:
0.8267 - val_loss: 0.1254 - val_acc: 0.8220
Epoch 28/30
42388/42388 [==============================] - 5s 115us/step - loss: 0.1235 - acc:
0.8265 - val_loss: 0.1250 - val_acc: 0.8236
Epoch 29/30
42388/42388 [==============================] - 5s 116us/step - loss: 0.1233 - acc:
0.8272 - val_loss: 0.1246 - val_acc: 0.8232
Epoch 30/30
42388/42388 [==============================] - 5s 118us/step - loss: 0.1232 - acc:
0.8270 - val_loss: 0.1246 - val_acc: 0.8233
```

In [399]:

```
#accuracy score
yhat = np.round(model.predict(X_ints_test + [X_test_num]))
print(mt.confusion_matrix(y_test,yhat), 'Accuracy ',mt.accuracy_score(y_test,yhat))
#my FN score
print('FN score ',custom_score(y_test,yhat))
```

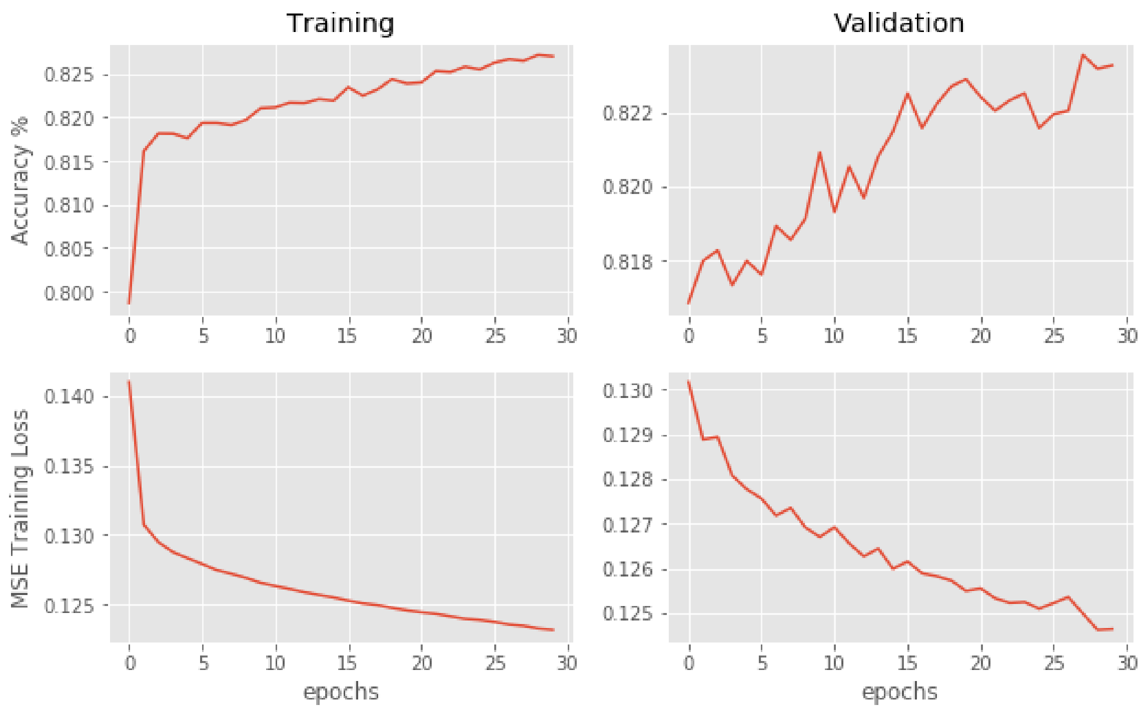
```
[[6646  731]
 [1139 2082]] Accuracy  0.8235516135119834
FN score  0.6090909090909091
```

In [400]:

```
from matplotlib import pyplot as plt

%matplotlib inline

plt.figure(figsize=(10,6))
plt.subplot(2,2,1)
plt.plot(history3.history['acc'])

plt.ylabel('Accuracy %')
plt.title('Training')
plt.subplot(2,2,2)
plt.plot(history3.history['val_acc'])
plt.title('Validation')

plt.subplot(2,2,3)
plt.plot(history3.history['loss'])
plt.ylabel('MSE Training Loss')
plt.xlabel('epochs')

plt.subplot(2,2,4)
plt.plot(history3.history['val_loss'])
plt.xlabel('epochs')
```

Out[400]:

```
Text(0.5, 0, 'epochs')
```



For train vs loss, I think it converge, the line reach the bottom. For validation, the line boucing around but it converge. In general, they converge. I think 30 epochs is fine.

In [450]:

```
#https://github.com/Thakugan/machine-learning-notebooks/blob/master/6-wide-and-deep-networks/mus
hroom-hunting.ipynb
num_folds = 5

acc_scores3 =[]
fn_scores3 = []

skf = StratifiedKFold(n_splits=num_folds, shuffle=True)
for i, (train, test) in enumerate(skf.split(X, y)):#here I used corss validation on my model

    model3, X_ints_train, X_ints_test, X_train_num = create_model3(X_train, X_test, X_train_num,
categorical_headers_ints, cross_columns2)

    model3.compile(optimizer='adagrad',
                  loss='mean_squared_error',
                  metrics=['accuracy'])

    model3.fit(X_ints_train + [X_train_num], y_train, epochs=30, batch_size=50, verbose=1)
#this is just what i do with out cross validation
    yhat = np.round(model3.predict(X_ints_test + [X_test_num]))
    print(mt.confusion_matrix(y_test,yhat))
    acc_score3 = mt.accuracy_score(y_test, yhat)
    acc_scores3.append(acc_score3)#append all acc for k-fold
    fn_score3 = custom_score(y_test,yhat)
    fn_scores3.append(fn_score3)#append all fn for k-fold
    print("Accuracy: ", acc_score3)
    print("Fn : ", fn_score3)
print(fn_scores3)
print(acc_scores3)
```

```
Epoch 1/30
42388/42388 [==============================] - 8s 194us/step - loss: 0.1387 - acc:
0.8043
Epoch 2/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1301 - acc:
0.8164
Epoch 3/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1289 - acc:
0.8174
Epoch 4/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1280 - acc:
0.8175
Epoch 5/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1272 - acc:
0.8187
Epoch 6/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1267 - acc:
0.8194
Epoch 7/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1263 - acc:
0.8214
Epoch 8/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1259 - acc:
0.8217
Epoch 9/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1255 - acc:
0.8220
Epoch 10/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1252 - acc:
0.8222
Epoch 11/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1249 - acc:
0.8234
Epoch 12/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1246 - acc:
0.8233
Epoch 13/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1243 - acc:
0.8238
Epoch 14/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1240 - acc:
0.8250
Epoch 15/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1238 - acc:
0.8240
Epoch 16/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1235 - acc:
0.8253
Epoch 17/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1233 - acc:
0.8250
Epoch 18/30
42388/42388 [==============================] - 5s 127us/step - loss: 0.1231 - acc:
0.8259
Epoch 19/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1228 - acc:
0.8268
Epoch 20/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1226 - acc:
0.8268
Epoch 21/30
```

```
42388/42388 [==============================] - 5s 127us/step - loss: 0.1223 - acc:
0.8269
Epoch 22/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1222 - acc:
0.8278
Epoch 23/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1220 - acc:
0.8279
Epoch 24/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1218 - acc:
0.8286
Epoch 25/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1216 - acc:
0.8284
Epoch 26/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1214 - acc:
0.8297
Epoch 27/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1213 - acc:
0.8293
Epoch 28/30
42388/42388 [==============================] - 5s 128us/step - loss: 0.1210 - acc:
0.8302
Epoch 29/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1209 - acc:
0.8300
Epoch 30/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1207 - acc:
0.8306
[[6554  823]
 [1071 2150]]
Accuracy:  0.8212870352896773
Fn :  0.5654699049630412
Epoch 1/30
42388/42388 [==============================] - 8s 198us/step - loss: 0.1375 - acc:
0.8037
Epoch 2/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1303 - acc:
0.8145
Epoch 3/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1292 - acc:
0.8164
Epoch 4/30
42388/42388 [==============================] - 6s 133us/step - loss: 0.1287 - acc:
0.8170
Epoch 5/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1282 - acc:
0.8161
Epoch 6/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1278 - acc:
0.8189
Epoch 7/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1274 - acc:
0.8189
Epoch 8/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1271 - acc:
0.8199
Epoch 9/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1267 - acc:
0.8204
Epoch 10/30
```

```
42388/42388 [==============================] - 6s 131us/step - loss: 0.1266 - acc:
0.8204
Epoch 11/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1262 - acc:
0.8215
Epoch 12/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1260 - acc:
0.8216
Epoch 13/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1257 - acc:
0.8225
Epoch 14/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1254 - acc:
0.8232
Epoch 15/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1252 - acc:
0.8234
Epoch 16/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1250 - acc:
0.8232
Epoch 17/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1247 - acc:
0.8233
Epoch 18/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1245 - acc:
0.8238
Epoch 19/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1244 - acc:
0.8236
Epoch 20/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1242 - acc:
0.8241
Epoch 21/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1241 - acc:
0.8243
Epoch 22/30
42388/42388 [==============================] - 6s 136us/step - loss: 0.1239 - acc:
0.8248
Epoch 23/30
42388/42388 [==============================] - 6s 139us/step - loss: 0.1238 - acc:
0.8249
Epoch 24/30
42388/42388 [==============================] - 6s 135us/step - loss: 0.1236 - acc:
0.8253
Epoch 25/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1234 - acc:
0.8254
Epoch 26/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1233 - acc:
0.8254
Epoch 27/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1232 - acc:
0.8262
Epoch 28/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1231 - acc:
0.8269
Epoch 29/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1230 - acc:
0.8259
Epoch 30/30
42388/42388 [==============================] - 6s 137us/step - loss: 0.1227 - acc:
```

```
0.8264
[[6658  719]
 [1172 2049]]
Accuracy:  0.8215701075674655
Fn :  0.6197778952934955
Epoch 1/30
42388/42388 [==============================] - 9s 206us/step - loss: 0.1380 - acc:
0.8036
Epoch 2/30
42388/42388 [==============================] - 6s 138us/step - loss: 0.1298 - acc:
0.8166
Epoch 3/30
42388/42388 [==============================] - 6s 134us/step - loss: 0.1287 - acc:
0.8181
Epoch 4/30
42388/42388 [==============================] - 6s 133us/step - loss: 0.1280 - acc:
0.8195
Epoch 5/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1275 - acc:
0.8201
Epoch 6/30
42388/42388 [==============================] - 6s 133us/step - loss: 0.1270 - acc:
0.8208
Epoch 7/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1266 - acc:
0.8218
Epoch 8/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1263 - acc:
0.8217
Epoch 9/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1260 - acc:
0.8222
Epoch 10/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1258 - acc:
0.8227
Epoch 11/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1254 - acc:
0.8229
Epoch 12/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1252 - acc:
0.8240
Epoch 13/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1249 - acc:
0.8239
Epoch 14/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1247 - acc:
0.8252
Epoch 15/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1245 - acc:
0.8240
Epoch 16/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1244 - acc:
0.8245
Epoch 17/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1241 - acc:
0.8250
Epoch 18/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1239 - acc:
0.8250
Epoch 19/30
42388/42388 [==============================] - 6s 133us/step - loss: 0.1237 - acc:
```

```
0.8250
Epoch 20/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1235 - acc:
0.8264
Epoch 21/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1233 - acc:
0.8262
Epoch 22/30
42388/42388 [==============================] - 6s 133us/step - loss: 0.1232 - acc:
0.8264
Epoch 23/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1229 - acc:
0.8271
Epoch 24/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1229 - acc:
0.8264
Epoch 25/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1227 - acc:
0.8264
Epoch 26/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1225 - acc:
0.8269
Epoch 27/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1224 - acc:
0.8275
Epoch 28/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1222 - acc:
0.8273
Epoch 29/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1221 - acc:
0.8272
Epoch 30/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1220 - acc:
0.8271
[[6757  620]
 [1241 1980]]
Accuracy:  0.8244008303453482
Fn :  0.6668457818377217
Epoch 1/30
42388/42388 [==============================] - 9s 201us/step - loss: 0.1375 - acc:
0.8062
Epoch 2/30
42388/42388 [==============================] - 6s 133us/step - loss: 0.1301 - acc:
0.8182
Epoch 3/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1290 - acc:
0.8197
Epoch 4/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1283 - acc:
0.8199
Epoch 5/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1277 - acc:
0.8214
Epoch 6/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1271 - acc:
0.8218
Epoch 7/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1268 - acc:
0.8230
Epoch 8/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1264 - acc:
```

```
0.8233
Epoch 9/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1261 - acc:
0.8243
Epoch 10/30
42388/42388 [==============================] - 5s 130us/step - loss: 0.1257 - acc:
0.8245
Epoch 11/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1254 - acc:
0.8255
Epoch 12/30
42388/42388 [==============================] - 5s 130us/step - loss: 0.1250 - acc:
0.8263
Epoch 13/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1246 - acc:
0.8260
Epoch 14/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1244 - acc:
0.8266
Epoch 15/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1241 - acc:
0.8270
Epoch 16/30
42388/42388 [==============================] - 5s 130us/step - loss: 0.1239 - acc:
0.8265
Epoch 17/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1237 - acc:
0.8270
Epoch 18/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1235 - acc:
0.8273
Epoch 19/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1234 - acc:
0.8283
Epoch 20/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1231 - acc:
0.8277
Epoch 21/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1229 - acc:
0.8275
Epoch 22/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1227 - acc:
0.8289
Epoch 23/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1225 - acc:
0.8287
Epoch 24/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1223 - acc:
0.8289
Epoch 25/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1221 - acc:
0.8299
Epoch 26/30
42388/42388 [==============================] - 5s 130us/step - loss: 0.1220 - acc:
0.8298
Epoch 27/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1218 - acc:
0.8299
Epoch 28/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1216 - acc:
0.8307
```

```
Epoch 29/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1213 - acc:
0.8308
Epoch 30/30
42388/42388 [==============================] - 5s 129us/step - loss: 0.1211 - acc:
0.8317
[[6647  730]
 [1141 2080]]
Accuracy:  0.823457256086054
Fn :  0.6098343132014965
Epoch 1/30
42388/42388 [==============================] - 9s 204us/step - loss: 0.1398 - acc:
0.8042
Epoch 2/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1302 - acc:
0.8151
Epoch 3/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1288 - acc:
0.8176
Epoch 4/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1279 - acc:
0.8178
Epoch 5/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1272 - acc:
0.8189
Epoch 6/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1268 - acc:
0.8204
Epoch 7/30
42388/42388 [==============================] - 6s 130us/step - loss: 0.1262 - acc:
0.8205
Epoch 8/30
42388/42388 [==============================] - 6s 134us/step - loss: 0.1260 - acc:
0.8216
Epoch 9/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1256 - acc:
0.8225
Epoch 10/30
42388/42388 [==============================] - 6s 134us/step - loss: 0.1253 - acc:
0.8223
Epoch 11/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1249 - acc:
0.8238
Epoch 12/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1247 - acc:
0.8240
Epoch 13/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1244 - acc:
0.8251
Epoch 14/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1241 - acc:
0.8255
Epoch 15/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1238 - acc:
0.8257
Epoch 16/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1235 - acc:
0.8266
Epoch 17/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1233 - acc:
0.8273
```

```
Epoch 18/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1231 - acc:
0.8276
Epoch 19/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1229 - acc:
0.8285
Epoch 20/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1227 - acc:
0.8292
Epoch 21/30
42388/42388 [==============================] - 6s 133us/step - loss: 0.1224 - acc:
0.8301
Epoch 22/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1222 - acc:
0.8301
Epoch 23/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1220 - acc:
0.8306
Epoch 24/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1218 - acc:
0.8315
Epoch 25/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1216 - acc:
0.8314
Epoch 26/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1214 - acc:
0.8326
Epoch 27/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1212 - acc:
0.8325
Epoch 28/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1210 - acc:
0.8330
Epoch 29/30
42388/42388 [==============================] - 6s 131us/step - loss: 0.1208 - acc:
0.8336
Epoch 30/30
42388/42388 [==============================] - 6s 132us/step - loss: 0.1206 - acc:
0.8340
[[6692  685]
 [1150 2071]]
Accuracy:  0.8268541234195131
Fn :  0.6267029972752044
[0.5654699049630412, 0.6197778952934955, 0.6668457818377217, 0.6098343132014965,
0.6267029972752044]
[0.8212870352896773, 0.8215701075674655, 0.8244008303453482, 0.823457256086054, 0.
8268541234195131]
```

Compare different models

Model1 layer3:

- Accuracy 0.8235516135119834
- FN score 0.6090909090909091

Model2 layer3:

- Accuracy 0.8243064729194187
- FN score 0.583780880773362

Model1 layer7:

- Accuracy 0.8259105491602189
- FN score 0.5934959349593496

Model2 layer7:

- Accuracy 0.8235516135119834
- FN score 0.6090909090909091

model1 kayer3 vs model2 layer3

In [482]:

```
#model1 kayer3 vs model2 layer3 acc
t = 2.26 / np.sqrt(10)
e = (1-np.array(acc_scores))-(1-np.array(acc_scores1))
stdtot =np.std(e)

dbar = np.mean(e)
print('model1 kayer3 vs model2 layer3 acc range :',dbar-t*stdtot,dbar+t*stdtot)
```

```
model1 kayer3 vs model2 layer3 acc range : -0.0020848617107025436 0.00272649220702
2611
```

Becasue the range is include 0, so we can not say that with 95% confident level, model1 kayer3 and model2 layer3 are statisitcally different base on accuracy.

In [484]:

```
#model1 kayer3 vs model2 layer3 fn
t = 2.26 / np.sqrt(10)
e = (1-np.array(fn_scores))-(1-np.array(fn_scores1))
stdtot =np.std(e)

dbar = np.mean(e)
print('model1 kayer3 vs model2 layer3 fn range :',dbar-t*stdtot,dbar+t*stdtot)
```

```
model1 kayer3 vs model2 layer3 fn range : -0.019409982824137216 0.0048360352343908
67
```

Becasue the range is include 0, so we can not say that with 95% confident level, model1 kayer3 and model2 layer3 are statisitcally different base on FN score.

In [488]:

```
print('Mean acc model1 3layer ',np.mean(np.array(acc_scores)))
print('Mean acc model2 3layer ',np.mean(np.array(acc_scores1)))
```

```
Mean acc model1 3layer  0.823287412719381
Mean acc model2 3layer  0.823608227967541
```

In [489]:

```
print('Mean fn model1 3layer ',np.mean(np.array(fn_scores)))
print('Mean fn model2 3layer ',np.mean(np.array(fn_scores1)))
```

```
Mean fn model1 3layer  0.5996149422379874
Mean fn model2 3layer  0.5923279684431142
```

Base on sttisitcal result, we can not say who is better than who. However, Both fn and acc of Model2 3layer is little bit better than medal1 3 layer. So I still think model2 3 layer is better.

model1 kayer7 vs model2 layer7

In [494]:

```
#model1 kayer7 vs model2 layer7 acc
t = 2.26 / np.sqrt(10)
e = (1-np.array(acc_scores2))-(1-np.array(acc_scores3))
stdtot =np.std(e)

dbar = np.mean(e)
print('model1 kayer7 vs model2 layer7 acc range :',dbar-t*stdtot,dbar+t*stdtot)
```

```
model1 kayer7 vs model2 layer7 acc range : -0.0011039088106156495 0.00355720188478
05803
```

Becasue the range is include 0, so we can not say that with 95% confident level, model1 kayer7 and model2 layer7 are statisitcally different base on accuracy score.

In [495]:

```
#model1 kayer7 vs model2 layer7 fn
t = 2.26 / np.sqrt(10)
e = (1-np.array(fn_scores2))-(1-np.array(fn_scores3))
stdtot =np.std(e)

dbar = np.mean(e)
print('model1 kayer7 vs model2 layer7 fn range :',dbar-t*stdtot,dbar+t*stdtot)
```

```
model1 kayer7 vs model2 layer7 fn range : -0.01795282656640379 0.02142611122104695
```

Becasue the range is include 0, so we can not say that with 95% confident level, model1 kayer7 and model2 layer7 are statisitcally different base on FN score.

In [496]:

```
print('Mean acc model1 7layer ',np.mean(np.array(acc_scores2)))
print('Mean acc model2 7layer ',np.mean(np.array(acc_scores3)))
```

```
Mean acc model1 7layer  0.8222872240045291
Mean acc model2 7layer  0.8235138705416116
```

In [497]:

```
print('Mean fn model1 7layer ',np.mean(np.array(fn_scores2)))
print('Mean fn model2 7layer ',np.mean(np.array(fn_scores3)))
```

```
Mean fn model1 7layer  0.6159895361868704
Mean fn model2 7layer  0.6177261785141919
```

Base on sttisitcal result, we can not say who is better than who. However, acc of Model2 7layer is little bit better than medal1 7 layer. And fn of model1 7layer is better. So I still think model2 7 layer is better because it have better accuracy.

model1 kayer3 vs model1 layer7

In [507]:

```
#model1 kayer3 vs model1 layer7 acc
t = 2.26 / np.sqrt(10)
e = (1-np.array(acc_scores))-(1-np.array(acc_scores2))
stdtot =np.std(e)

dbar = np.mean(e)
print('model1 kayer3 vs model1 layer7 acc range :',dbar-t*stdtot,dbar+t*stdtot)
```

```
model1 kayer3 vs model1 layer7 acc range : -0.002823974713825091 0.000823597284121
4207
```

Becasue the range is include 0, so we can not say that with 95% confident level, model1 kayer3 and model1 layer7 are statisitcally different base on accuracy score.

In [508]:

```
#model1 kayer3 vs model1 layer7 fn
t = 2.26 / np.sqrt(10)
e = (1-np.array(fn_scores))-(1-np.array(fn_scores2))
stdtot =np.std(e)

dbar = np.mean(e)
print('model1 kayer3 vs model1 layer7 fn range :',dbar-t*stdtot,dbar+t*stdtot)
```

```
model1 kayer3 vs model1 layer7 fn range : 0.0026688743423036836 0.0300803135554621
15
```

Becasue the range is not include 0, so we can say that with 95% confident level, model1 kayer3 and model1 layer7 are statisitcally different base on FN score.

In [501]:

```
print('Mean acc model1 3layer ',np.mean(np.array(acc_scores)))
print('Mean acc model2 7layer ',np.mean(np.array(acc_scores2)))
```

```
Mean acc model1 3layer  0.823287412719381
Mean acc model2 7layer  0.8222872240045291
```

In [502]:

```
print('Mean fn model1 3layer ',np.mean(np.array(fn_scores)))
print('Mean fn model2 7layer ',np.mean(np.array(fn_scores2)))
```

```
Mean fn model1 3layer  0.5996149422379874
Mean fn model2 7layer  0.6159895361868704
```

Base on sttisitcal result of FN, we can say they are different. Both acc and FN of Model1 3layer is little bit better than medal1 7 layer. So I think model1 3 layer is better.

model2 kayer3 vs model2 layer7

In [503]:

```
#model2 kayer3 vs model2 layer7 acc
t = 2.26 / np.sqrt(10)
e = (1-np.array(acc_scores1))-(1-np.array(acc_scores3))
stdtot =np.std(e)

dbar = np.mean(e)
print('model2 kayer3 vs model2 layer7 acc range :',dbar-t*stdtot,dbar+t*stdtot)
```

```
model2 kayer3 vs model2 layer7 acc range : -0.0017522132213550997 0.00156349836949
62931
```

Becasue the range is include 0, so we can not say that with 95% confident level, model2 kayer3 and model2 layer7 are statisitcally different base on accuracy score.

In [504]:

```
#model1 kayer7 vs model2 layer7 fn
t = 2.26 / np.sqrt(10)
e = (1-np.array(fn_scores1))-(1-np.array(fn_scores3))
stdtot =np.std(e)

dbar = np.mean(e)
print('model2 kayer3 vs model2 layer7 fn range :',dbar-t*stdtot,dbar+t*stdtot)
```

```
model2 kayer3 vs model2 layer7 fn range : -0.011572829058021553 0.0623692492001768
6
```

Becasue the range is include 0, so we can not say that with 95% confident level, model2 kayer3 and model2 layer7 are statisitcally different base on FN score.

In [505]:

```
print('Mean acc model2 3layer ',np.mean(np.array(acc_scores1)))
print('Mean acc model2 7layer ',np.mean(np.array(acc_scores3)))
```

```
Mean acc model2 3layer  0.823608227967541
Mean acc model2 7layer  0.8235138705416116
```

In [506]:

```
print('Mean fn model2 3layer ',np.mean(np.array(fn_scores1)))
print('Mean fn model2 7layer ',np.mean(np.array(fn_scores3)))
```

```
Mean fn model2 3layer  0.5923279684431142
Mean fn model2 7layer  0.6177261785141919
```

Base on sttisitcal result, we can not say they are different. Both acc and FN of Model2 3layer is little bit better than medal2 7 layer. So I think model2 3 layer is better.

Overall, base on 4 comparision, Model2 layer3 is the best I think.

MLP

In [469]:

```
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
X_train_ohe = ohe.fit_transform(df_train[categorical_headers_ints].values)
X_test_ohe = ohe.transform(df_test[categorical_headers_ints].values)

# and save off the numeric features
X_train_num =  df_train[numeric_headers].values
X_test_num = df_test[numeric_headers].values
```

In [470]:

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(50,),
                       solver='sgd',
                       learning_rate_init=0.01,
                       max_iter=100,
                       random_state=1)
```

In [471]:

```
#https://stackoverflow.com/questions/25453173/numpy-array-concatenation-error-0-d-arrays-cant-be
-concatenated
#np.concatenate() doesn't work with sparse matrices. i use np.concatenate([B.todense(),A]).
x_train = np.concatenate((X_train_ohe.todense(), X_train_num), axis=1)#concatenate categorical v
ariables and numeric variables
y_train = y_train
x_test = np.concatenate((X_test_ohe.todense(), X_test_num), axis=1)
y_test = y_test
```

In [475]:

```
%%time
mlp.fit(x_train, y_train)
#accuracy score
yhat = mlp.predict(x_test)
print(mt.confusion_matrix(y_test,yhat), 'Accuracy ',mt.accuracy_score(y_test,yhat))
#my FN score
print('FN score ',custom_score(y_test,yhat))
```

```
[[6711  595]
 [1339 1953]] Accuracy  0.8175127382525005
FN score  0.7034675169390195
Wall time: 16.7 s
```

In [478]:

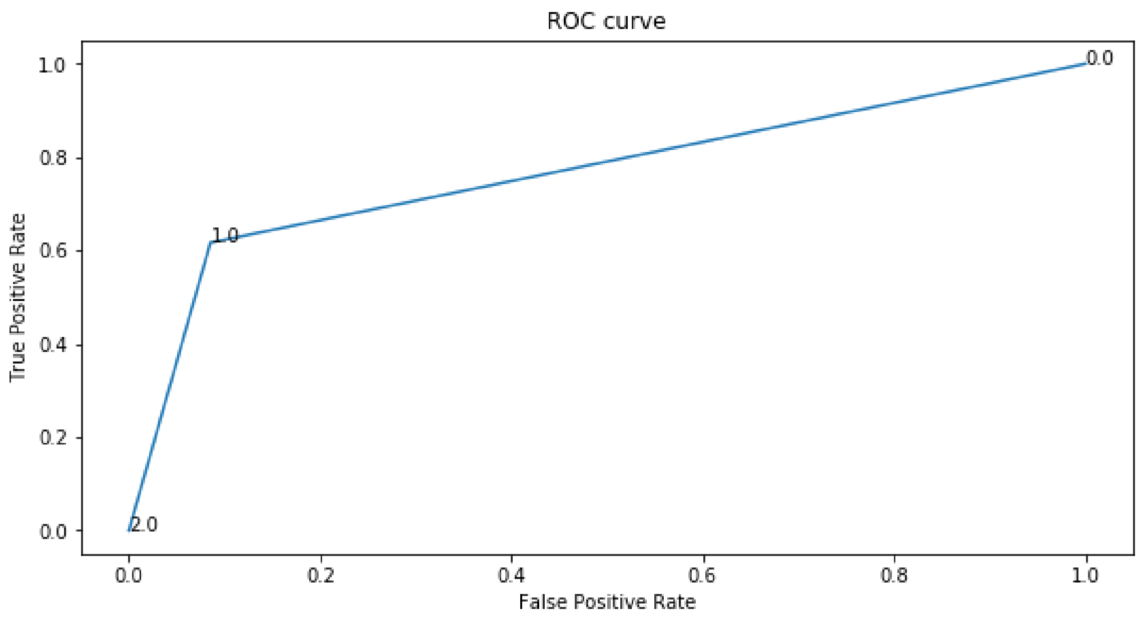
```
acc_scores4 = []
fn_scores4 = []

skf = StratifiedKFold(n_splits=num_folds, shuffle=True)
for i, (train, test) in enumerate(skf.split(X, y)):

    mlp.fit(x_train, y_train)

    yhat = mlp.predict(x_test)
    print(mt.confusion_matrix(y_test,yhat))
    acc_score4 = mt.accuracy_score(y_test, yhat)
    acc_scores4.append(acc_score4)#append all acc for k-fold
    fn_score4 = custom_score(y_test,yhat)
    fn_scores4.append(fn_score4)#append all fn for k-fold
    print("Accuracy: ", acc_score4)
    print("Fn : ", fn_score4)
print(fn_scores4)
print(acc_scores4)
```

```
[[6711  595]
 [1339 1953]]
Accuracy:  0.8175127382525005
Fn :  0.6923474663908997
[[6711  595]
 [1339 1953]]
Accuracy:  0.8175127382525005
Fn :  0.6923474663908997
[[6711  595]
 [1339 1953]]
Accuracy:  0.8175127382525005
Fn :  0.6923474663908997
[[6711  595]
 [1339 1953]]
Accuracy:  0.8175127382525005
Fn :  0.6923474663908997
[[6711  595]
 [1339 1953]]
Accuracy:  0.8175127382525005
Fn :  0.6923474663908997
[0.6923474663908997, 0.6923474663908997, 0.6923474663908997, 0.6923474663908997,
0.6923474663908997]
[0.8175127382525005, 0.8175127382525005, 0.8175127382525005, 0.8175127382525005,
0.8175127382525005]
```

In [41]:

```
#I used this on ICA1 and just copy from my old work
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
from sklearn.metrics import roc_curve, roc_auc_score
import numpy as np
import matplotlib.pyplot
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_test, yhat)
threshold = np.round(thresholds, decimals=2)


plt.figure(figsize=(10,5))
plt.plot(fpr,tpr)
labels = [threshold[i] for i in range(3)]
for i in range (3):
    xy=(fpr[i],tpr[i])
    plt.annotate(labels[i],xy)

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve')
print('Area under the curve(AUC) is ',roc_auc_score(y_test, yhat))
```

```
Area under the curve(AUC) is  0.7658001681387965
```



For this one, from roc curve, I calculate the area under curve is 0.76.

model2 layer3

In [42]:

```
# lets also add the history variable to see how we are doing
# and lets add a validation set to keep track of our progress
history1 = model.fit(X_ints_train+ [X_train_num],
                    y_train,
                    epochs=20,
                    batch_size=50,
                    verbose=1,
                    validation_data = (X_ints_test + [X_test_num], y_test))
```

```
Train on 42388 samples, validate on 10598 samples
Epoch 1/20
42388/42388 [==============================] - 5s 120us/step - loss: 0.1230 - acc:
0.8263 - val_loss: 0.1223 - val_acc: 0.8278
Epoch 2/20
42388/42388 [==============================] - 5s 118us/step - loss: 0.1228 - acc:
0.8260 - val_loss: 0.1218 - val_acc: 0.8286
Epoch 3/20
42388/42388 [==============================] - 5s 117us/step - loss: 0.1226 - acc:
0.8269 - val_loss: 0.1219 - val_acc: 0.8282
Epoch 4/20
42388/42388 [==============================] - 5s 117us/step - loss: 0.1224 - acc:
0.8273 - val_loss: 0.1219 - val_acc: 0.8275
Epoch 5/20
42388/42388 [==============================] - 5s 118us/step - loss: 0.1222 - acc:
0.8273 - val_loss: 0.1216 - val_acc: 0.8299
Epoch 6/20
42388/42388 [==============================] - 5s 125us/step - loss: 0.1220 - acc:
0.8276 - val_loss: 0.1216 - val_acc: 0.8292
Epoch 7/20
42388/42388 [==============================] - 5s 123us/step - loss: 0.1219 - acc:
0.8277 - val_loss: 0.1215 - val_acc: 0.8287
Epoch 8/20
42388/42388 [==============================] - 5s 117us/step - loss: 0.1217 - acc:
0.8294 - val_loss: 0.1215 - val_acc: 0.8285
Epoch 9/20
42388/42388 [==============================] - 5s 117us/step - loss: 0.1215 - acc:
0.8297 - val_loss: 0.1212 - val_acc: 0.8291
Epoch 10/20
42388/42388 [==============================] - 5s 115us/step - loss: 0.1213 - acc:
0.8293 - val_loss: 0.1212 - val_acc: 0.8281
Epoch 11/20
42388/42388 [==============================] - 5s 119us/step - loss: 0.1211 - acc:
0.8303 - val_loss: 0.1212 - val_acc: 0.8300
Epoch 12/20
42388/42388 [==============================] - 5s 116us/step - loss: 0.1210 - acc:
0.8303 - val_loss: 0.1212 - val_acc: 0.8288
Epoch 13/20
42388/42388 [==============================] - 5s 112us/step - loss: 0.1208 - acc:
0.8312 - val_loss: 0.1213 - val_acc: 0.8286
Epoch 14/20
42388/42388 [==============================] - 5s 112us/step - loss: 0.1206 - acc:
0.8317 - val_loss: 0.1210 - val_acc: 0.8286
Epoch 15/20
42388/42388 [==============================] - 5s 113us/step - loss: 0.1204 - acc:
0.8320 - val_loss: 0.1215 - val_acc: 0.8259
Epoch 16/20
42388/42388 [==============================] - 5s 118us/step - loss: 0.1203 - acc:
0.8319 - val_loss: 0.1209 - val_acc: 0.8290
Epoch 17/20
42388/42388 [==============================] - 5s 117us/step - loss: 0.1202 - acc:
0.8325 - val_loss: 0.1210 - val_acc: 0.8282
Epoch 18/20
42388/42388 [==============================] - 5s 112us/step - loss: 0.1200 - acc:
0.8325 - val_loss: 0.1212 - val_acc: 0.8295
Epoch 19/20
42388/42388 [==============================] - 5s 112us/step - loss: 0.1199 - acc:
0.8320 - val_loss: 0.1209 - val_acc: 0.8293
Epoch 20/20
42388/42388 [==============================] - 5s 112us/step - loss: 0.1197 - acc:
0.8325 - val_loss: 0.1208 - val_acc: 0.8294
```

In [43]:

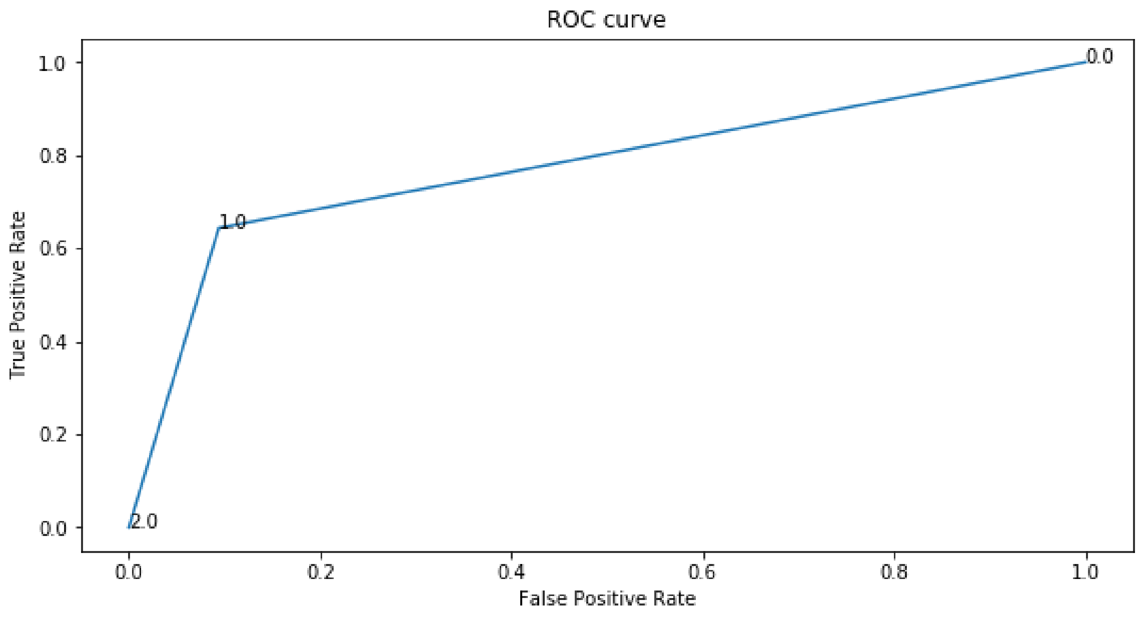
```
#accuracy score
yhat = np.round(model.predict(X_ints_test + [X_test_num]))
print(mt.confusion_matrix(y_test,yhat), 'Accuracy ',mt.accuracy_score(y_test,yhat))
#my FN score
print('FN score ',custom_score(y_test,yhat))
```

```
[[6800  705]
 [1103 1990]] Accuracy  0.8294017739196075
FN score  0.610066371681416
```

In [44]:

```
#I used this on ICA1 and just copy from my old work
#https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html
from sklearn.metrics import roc_curve, roc_auc_score
import numpy as np
import matplotlib.pyplot
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_test, yhat)
threshold = np.round(thresholds, decimals=2)


plt.figure(figsize=(10,5))
plt.plot(fpr,tpr)
labels = [threshold[i] for i in range(3)]
for i in range (3):
    xy=(fpr[i],tpr[i])
    plt.annotate(labels[i],xy)

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve')
print('Area under the curve(AUC) is ',roc_auc_score(y_test, yhat))
```

```
Area under the curve(AUC) is  0.7747254605346625
```



For this one, from roc curve, I calculate the area under curve is 0.77.

The AUC of mlp is litter bit lower than my model, the accuracy is really close. Fn score of my model is 3% better than mlp. I think my model is little better base on the score and AUC.

In [490]:

```
#model2 kayer3 vs mlp
t = 2.26 / np.sqrt(10)
e = (1-np.array(acc_scores1))-(1-np.array(acc_scores4))
stdtot =np.std(e)

dbar = np.mean(e)
print('model2 layer3 vs mlp acc range :',dbar-t*stdtot,dbar+t*stdtot)
```

```
model2 layer3 vs mlp acc range : -0.007178318195991879 -0.005012661234089216
```

Becasue the range is not include 0, so we can say that with 95% confident level, model2 layer3 and mlp are statisitcally different base on acc score.

In [491]:

```
#model2 kayer3 vs mlp
t = 2.26 / np.sqrt(10)
e = (1-np.array(fn_scores1))-(1-np.array(fn_scores4))
stdtot =np.std(e)

dbar = np.mean(e)
print('model2 layer3 vs mlp fn range :',dbar-t*stdtot,dbar+t*stdtot)
```

```
model2 layer3 vs mlp fn range : 0.08386956769253306 0.1161694282030388
```

Becasue the range is not include 0, so we can say that with 95% confident level, model2 layer3 and mlp are statisitcally different base on FN score.

In [492]:

```
print('Mean acc model2 3layer ',np.mean(np.array(acc_scores1)))
print('Mean acc mlp ',np.mean(np.array(acc_scores4)))
```

```
Mean acc model2 3layer  0.823608227967541
Mean acc mlp  0.8175127382525005
```

In [493]:

```
print('Mean fn model2 3layer ',np.mean(np.array(fn_scores1)))
print('Mean fn mlp ',np.mean(np.array(fn_scores4)))
```

```
Mean fn model2 3layer  0.5923279684431142
Mean fn mlp  0.6923474663908997
```

By comparing the accuracy, FN score and statistical method, my model is better than mlp.

Exceptional Work

In [19]:

```
from keras.layers import concatenate
from sklearn.preprocessing import OneHotEncoder
#model2
#'HEAT_D','AC','QUALIFIED','STRUCT_D','CNDTN_D','ROOF_D','INTWALL_D'
#"BATHRM","BEDRM","AYB_year","YR_RMDL_year","EYB_year","STORAGE","GBA","KITCHENS"
cross_columns = [['HEAT_D','INTWALL_D'],
                 ['AC', 'ROOF_D'],
                 ['ROOF_D','INTWALL_D','STRUCT_D']]


# and save off the numeric features
X_train_num =  df_train[numeric_headers].values
X_test_num = df_test[numeric_headers].values


# we need to create separate lists for each branch
embed_branches = []
X_ints_train = []
X_ints_test = []
all_inputs = []
all_wide_branch_outputs = []

for cols in cross_columns:
    # encode crossed columns as ints for the embedding
    enc = LabelEncoder()

    # create crossed labels
    X_crossed_train = df_train[cols].apply(lambda x: '_'.join(x), axis=1)
    X_crossed_test = df_test[cols].apply(lambda x: '_'.join(x), axis=1)

    enc.fit(np.hstack((X_crossed_train.values,  X_crossed_test.values)))
    X_crossed_train = enc.transform(X_crossed_train)
    X_crossed_test = enc.transform(X_crossed_test)
    X_ints_train.append( X_crossed_train )
    X_ints_test.append( X_crossed_test )

    # get the number of categories
    N = max(X_ints_train[-1]+1) # same as the max(df_train[col])

    # create embedding branch from the number of categories
    inputs = Input(shape=(1,),dtype='int32', name = '_'.join(cols))
    all_inputs.append(inputs)
    x = Embedding(input_dim=N,
                  output_dim=int(np.sqrt(N)),
                  input_length=1, name = '_'.join(cols)+'_embed')(inputs)
    x = Flatten()(x)
    all_wide_branch_outputs.append(x)

# merge the branches together
wide_branch = concatenate(all_wide_branch_outputs, name='wide_concat')
wide_branch = Dense(units=1,activation='sigmoid',name='wide_combined')(wide_branch)

# reset this input branch
all_deep_branch_outputs = []
# add in the embeddings
for col in categorical_headers_ints:
    # encode as ints for the embedding
    X_ints_train.append( df_train[col].values )
    X_ints_test.append( df_test[col].values )
    # get the number of categories
    N = max(X_ints_train[-1]+1) # same as the max(df_train[col])

    # create embedding branch from the number of categories
    inputs = Input(shape=(1,),dtype='int32', name=col)
    all_inputs.append(inputs)
    x = Embedding(input_dim=N,
                  output_dim=int(np.sqrt(N)),
                  input_length=1, name=col+'_embed')(inputs)
    x = Flatten()(x)
    all_deep_branch_outputs.append(x)

# also get a dense branch of the numeric features
all_inputs.append(Input(shape=(X_train_num.shape[1],),
                        sparse=False,
                        name='numeric_data'))

x = Dense(units=20, activation='relu',name='numeric_1')(all_inputs[-1])
all_deep_branch_outputs.append( x )

# merge the deep branches together
deep_branch = concatenate(all_deep_branch_outputs,name='concat_embeds')
deep_branch = Dense(units=50,activation='relu', name='deep1')(deep_branch)
deep_branch = Dense(units=25,activation='relu', name='deep2')(deep_branch)
deep_branch = Dense(units=10,activation='relu', name='deep3')(deep_branch)

final_branch = concatenate([wide_branch, deep_branch],name='concat_deep_wide')
final_branch = Dense(units=1,activation='sigmoid',name='combined')(final_branch)

model = Model(inputs=all_inputs, outputs=final_branch)
```

In [20]:

```
model.compile(optimizer='adagrad',
              loss='mean_squared_error',
              metrics=['accuracy'])
```

In [21]:

```
# lets also add the history variable to see how we are doing
# and lets add a validation set to keep track of our progress
history1 = model.fit(X_ints_train+ [X_train_num],
                    y_train,
                    epochs=20,
                    batch_size=50,
                    verbose=1,
                    validation_data = (X_ints_test + [X_test_num], y_test))
```

```
Train on 42388 samples, validate on 10598 samples
Epoch 1/20
42388/42388 [==============================] - 5s 110us/step - loss: 0.1359 - acc:
0.8078 - val_loss: 0.1336 - val_acc: 0.8096
Epoch 2/20
42388/42388 [==============================] - 4s 101us/step - loss: 0.1292 - acc:
0.8164 - val_loss: 0.1322 - val_acc: 0.8096
Epoch 3/20
42388/42388 [==============================] - 4s 101us/step - loss: 0.1281 - acc:
0.8182 - val_loss: 0.1315 - val_acc: 0.8102
Epoch 4/20
42388/42388 [==============================] - 4s 100us/step - loss: 0.1273 - acc:
0.8188 - val_loss: 0.1310 - val_acc: 0.8128
Epoch 5/20
42388/42388 [==============================] - 4s 102us/step - loss: 0.1268 - acc:
0.8202 - val_loss: 0.1305 - val_acc: 0.8140
Epoch 6/20
42388/42388 [==============================] - 4s 101us/step - loss: 0.1264 - acc:
0.8202 - val_loss: 0.1306 - val_acc: 0.8119
Epoch 7/20
42388/42388 [==============================] - 4s 101us/step - loss: 0.1260 - acc:
0.8208 - val_loss: 0.1305 - val_acc: 0.8128
Epoch 8/20
42388/42388 [==============================] - 4s 101us/step - loss: 0.1257 - acc:
0.8215 - val_loss: 0.1298 - val_acc: 0.8133
Epoch 9/20
42388/42388 [==============================] - 4s 102us/step - loss: 0.1254 - acc:
0.8225 - val_loss: 0.1296 - val_acc: 0.8131
Epoch 10/20
42388/42388 [==============================] - 4s 101us/step - loss: 0.1251 - acc:
0.8227 - val_loss: 0.1294 - val_acc: 0.8138
Epoch 11/20
42388/42388 [==============================] - 4s 102us/step - loss: 0.1249 - acc:
0.8242 - val_loss: 0.1293 - val_acc: 0.8151
Epoch 12/20
42388/42388 [==============================] - 4s 102us/step - loss: 0.1247 - acc:
0.8233 - val_loss: 0.1294 - val_acc: 0.8147
Epoch 13/20
42388/42388 [==============================] - 4s 101us/step - loss: 0.1245 - acc:
0.8247 - val_loss: 0.1289 - val_acc: 0.8136
Epoch 14/20
42388/42388 [==============================] - 4s 101us/step - loss: 0.1243 - acc:
0.8249 - val_loss: 0.1289 - val_acc: 0.8142
Epoch 15/20
42388/42388 [==============================] - 4s 102us/step - loss: 0.1241 - acc:
0.8253 - val_loss: 0.1288 - val_acc: 0.8144
Epoch 16/20
42388/42388 [==============================] - 4s 101us/step - loss: 0.1240 - acc:
0.8251 - val_loss: 0.1289 - val_acc: 0.8142
Epoch 17/20
42388/42388 [==============================] - 4s 101us/step - loss: 0.1238 - acc:
0.8255 - val_loss: 0.1289 - val_acc: 0.8129
Epoch 18/20
42388/42388 [==============================] - 4s 101us/step - loss: 0.1236 - acc:
0.8261 - val_loss: 0.1286 - val_acc: 0.8145
Epoch 19/20
42388/42388 [==============================] - 4s 101us/step - loss: 0.1235 - acc:
0.8267 - val_loss: 0.1285 - val_acc: 0.8145
Epoch 20/20
42388/42388 [==============================] - 4s 101us/step - loss: 0.1233 - acc:
0.8275 - val_loss: 0.1284 - val_acc: 0.8138
```

In [149]:

```
#https://github.com/Thakugan/machine-learning-notebooks/blob/master/6-wide-and-deep-networks/mus
hroom-hunting.ipynb
#by using for loop, I can get all categorical variables
weights_int = []
for ind in model.layers:
    if '_int' in ind.name:
        print(ind.name)#this will return me all categorical variables with both embed and int, w
hat we want is embed
        weights_int.append(ind.get_weights())#append all weight from those categorical variables
```

```
HEAT_D_int
AC_int
QUALIFIED_int
STRUCT_D_int
CNDTN_D_int
ROOF_D_int
INTWALL_D_int
HEAT_D_int_embed
AC_int_embed
QUALIFIED_int_embed
STRUCT_D_int_embed
CNDTN_D_int_embed
ROOF_D_int_embed
INTWALL_D_int_embed
```

In [303]:

```
#ROOF_D is what I am interested
np.array(weights_int)[12]
```

Out[303]:

```
[array([[ 0.0251433 ,  0.0002808 ,  0.02129444],
        [-0.09512077,  0.00110211,  0.07112268],
        [ 0.07666893,  0.10092183, -0.20189363],
        [ 0.06469727, -0.01014414, -0.1404636 ],
        [-0.0499163 , -0.07063429,  0.07669131],
        [-0.17171095, -0.04373584,  0.10220618],
        [ 0.13047557,  0.16382605, -0.1913268 ],
        [-0.09523301,  0.0584864 ,  0.18685316],
        [-0.15185204, -0.03680865,  0.14271015],
        [ 0.02024156,  0.03793027,  0.00191131],
        [-0.06693493,  0.01291882,  0.04637688],
        [-0.16931182, -0.18953107,  0.0643701 ],
        [ 0.04365172, -0.05617747, -0.16290416],
        [ 0.10547858,  0.1094261 ,  0.1015405 ],
        [-0.04352704, -0.06455085,  0.05822873]], dtype=float32)]
```

In [310]:

```
dftsne = np.array(weights_int[12]).reshape(15,3)
Label_name = pd.Series(df_train_n.ROOF_D).unique()#get the label for roof
```

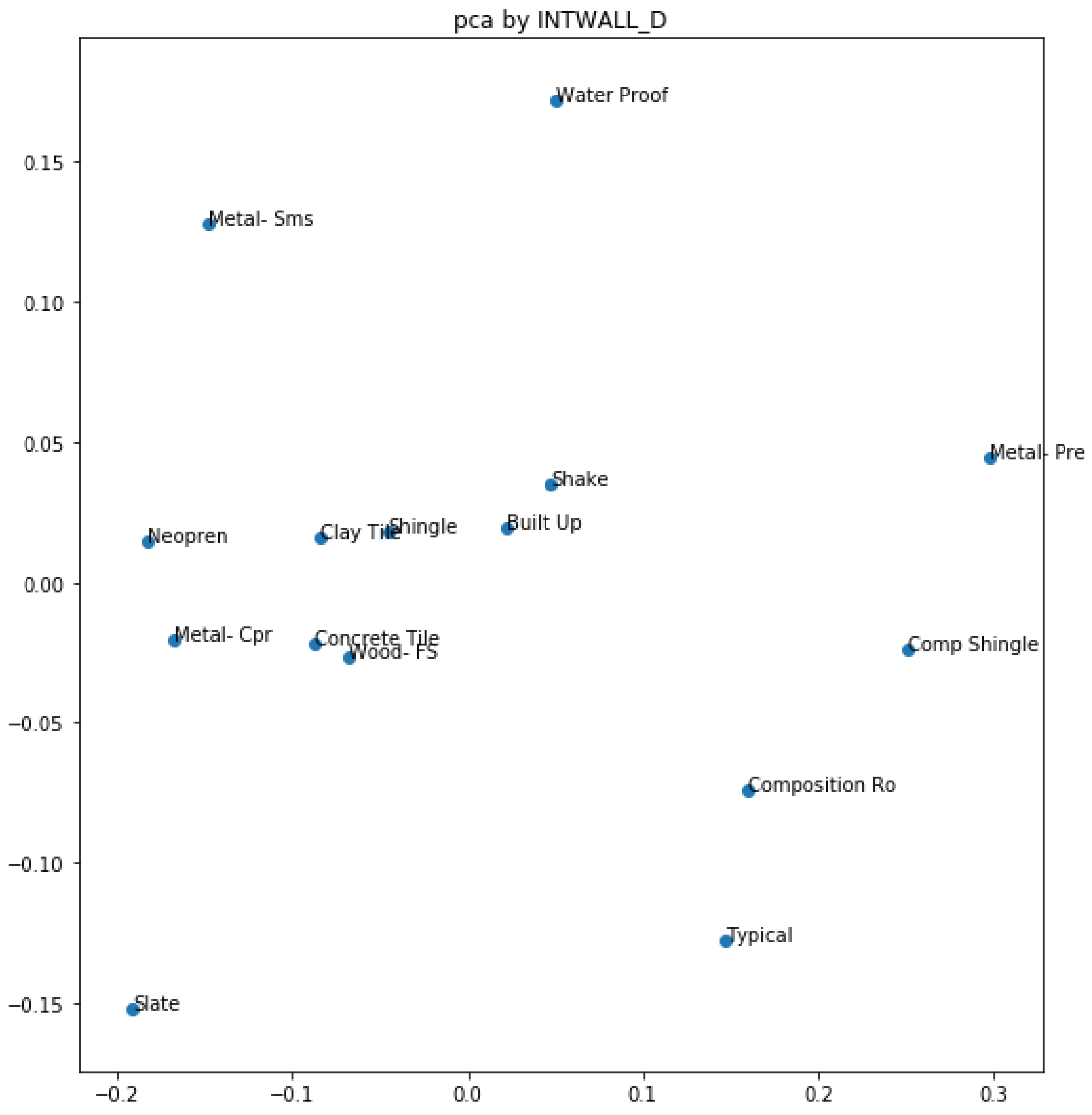
In [299]:

```
#http://alexanderfabisch.github.io/t-sne-in-scikit-learn.html
import sklearn
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
```

In [320]:

```
#I used pca to reduce demention to 2 dim
pca = PCA(n_components=2)
pca1 = pca.fit_transform(dftsne)
plt.figure(figsize=(20, 10))
plt.subplot(1,2,1)
#plot the 2dim
plt.scatter(pca1[:, 0], pca1[:, 1])

labels = [np.sort(Label_name)[i] for i in range(len(np.sort(Label_name)))]
#find out that python order categorical values by alphabet
#label the point, i used this in ica1
for i in range (len(np.sort(Label_name))):
    xy=(pca1[:, 0][i],pca1[:, 1][i])
    plt.annotate(labels[i],xy)


plt.title('pca by INTWALL_D')
```

```
Out[320]:

Text(0.5, 1.0, 'pca by INTWALL_D')
```


pca by INTWALL_D
Water Proof
Metal- Sms
Metal- Pre
Shake
Neopren
Clay Tile
Shingle
Built Up
Metal- Cpr
Concrete Tile
Wood- FS
Comp Shingle
Composition Ro
Typical
Slate
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.2
−0.1
0.0
0.1
0.2
0.3

We can see that shake and build up kind of similar input because they are close, but little bit different.

Clay tile and shingle are also similar input with bit different.

Concrete tile and wood-fs are really similar with only little-little bit different.

Neopren and metal-cpr are kind of group with clay tile and concrete, but with same distance between each other.

For above 4 group, they are tend to be a big group.

Metal-pre, comp shingle, composition ro and typical are also keep not close distance one by one. For this group, they are differnt from other big group.

Metal-sms, slate water proff are outside, not too close to any other point, they keep far enough distance to other points. They are not touch with any other big or small group.