2009 FIES: Predicting Poverty Index Using Deep Neural Network

Author: Cennen Del Rosario
Date: February 20, 2020

SHOW CODE

A. Logistic Regression Model as Benchmark Model

In this part, we will use python to make another baseline model for our analysis using Logistic Regression.
To start, we load the data and do some pre-processing.

```
GOOGLE_DRIVE_STAT219_PATH = 'drive/My Drive/Colab Notebooks/Stat 219'
POVERTY_CUT_OFF = 1403.00
SEED_NUMBER = 100

data_df = pd.read_csv(GOOGLE_DRIVE_STAT219_PATH + "/data/FIES_6.csv")
data_df['mon_per_cap_inc'] = data_df['toinc'] / (12 * data_df['z2101_tot_mem'])
data_df['poverty'] = (data_df['mon_per_cap_inc'] <= POVERTY_CUT_OFF) * 1

data_df['fsize'] = data_df['fsize'].replace(115, 11.5)
data_df.iloc[:5,:10]
```

	Unnamed: 0	w_regn	w_prv	w_id_recode	w_shsn	w_hcn	w_urb2	w_str2	w_psu	w_rep
0	1065	6	4	401011000	1	6	Urban	21022	7501	2
1	1066	6	4	401011000	2	43	Urban	21022	7501	2
2	1067	6	4	401011000	3	82	Urban	21022	7501	2
3	1068	6	4	401011000	4	120	Urban	21022	7501	2
4	1069	6	4	401011000	5	158	Urban	21022	7501	2

```
data_df = data_df.dropna(axis=1).drop_duplicates()
```

1. Separate and Encode Variables

For the purpose of this analysis, and as we will see later for the alternative Deep Neural Network model, we identified important variables and assigned them into 8 groups:

1. **Inflows:** these are variables that indicate flow of money towards the household/family. These include but are not limited to receivables, wages, etc.
2. **Expenses:** all variables that indicate expenditures in food, services, disbursments, etc.
3. **Outflows:** the opposite of inflows. This indicates outward flow of money other than expenditures. Examples are loan payments, insurance premiums, etc.
4. **Taxes:** variables related to tax payments and government fees.
5. **Family:** variables that pertain to household head and household members.
6. **House:** variables that are related to items found or used by the household. It includes but is not limited to number of appliances or building types.
7. **Geography:** these are province, urbanity and strata.
8. **Others:** other important factors and indicators.

```
columns = data_df.columns[1:]
numeric_columns = data_df._get_numeric_data().columns
categorical_columns = list(set(columns) - set(numeric_columns))
main_variables = ['torec','wages','othin','totex','taxes','fsize','z2101_tot_mem']
output_variables = ['toinc', 'poverty']
```

```
inflow_variables = ['torec','wages','othin','netsh','conab','condo','rentl','intrs',
                    'pnsns','dvdnd','ifams','regft','osinc','eainc','eacfggrs',
                    'ealprgrs','eafisgrs','eaforgrs','eatrdgrs','eamfggrs','eacpsgrs',
                    'eatcsgrs','eamnggrs','eacongrs','eanecgrs','eacfg','ealpr','eafis',
                    'eafor','eatrd','eamfg','eacps','eatcs','eamng','eacon','eanec',
```

```
                     'eafor','eatrd','eamfg','eacps','eatcs','eamng','eacon','eanec',
                    'winng','prfit','bkpay','inhrt','othre','aginc','nagin']
inflow_variables_cat = list(set(inflow_variables) - set(numeric_columns))
print('Inflow categorical variables:')
print(inflow_variables_cat)

expense_variables = ['totex','food','creal','trice','tcorn','bread','bisct','flour',
                     'ncake','nudle','ocrep','roots','ptato','casva','cmote','tgabi',
                     'otrut','fruit','frfrt','leveg','frveg','beans','otveg','ocrop',
                     'frpre','vgpre','otpre','meat','fchic','fbeef','fpork','otfmt',
                     'canmt','uncmt','dairy','tmilk','conds','evapo','powdr','fresh',
                     'icrem','otdry','teggs','feggs','peggs','fishm','ffish','cnfsh',
                     'drfsh','slfsh','otmrn','cofct','cofee','cofpr','cofbn','cocoa',
                     'coapt','coapr','coabn','tea','teapr','tealv','nonal','carbd',
                     'ncarb','othdr','botle','fdnec','sugar','sugpr','ckoil',
                     'margn','sauce','tsalt','otspc','mlout','ofnec','fhome','fdout',
                     'mlsch','mlwrk','mlres','snack','albev','tbeer','nwine','otbev',
                     'nfood','tbcco','cigrt','cigar','ottob','fuel','a1022','a1032',
                     'a1042','a1052','a1062','a1072','a1082','a1092','trcom','a2022',
                     'a2032','a2042','a2052','a2062','a2072','a2082','a2092','a2102',
                     'a2112','a2122','a2132','hoper','a4022','a4032','a4042','a4052',
                     'a4062','a4072','a4082','a4092','a4112','a4122','dserv','a4132',
                     'a4142','a4152','a4162','prcre','cloth','educ','rcrtn','medic',
                     'ndfur','house','rpair','occsn','gftot','othex','otdis','totdi',
                     'eacfgexp','ealprexp','eafisexp','eaforexp','eatrdexp','eamfgexp',
                     'eacpsexp','eatcsexp','eamngexp','eaconexp','eanecexp']
expense_variables_cat = list(set(expense_variables) - set(numeric_columns))
print('Expense categorical variables:')
print(expense_variables_cat)


outflow_variables = ['b8072','b8082','b8092','b9022','b9032','b9042','b9052','b9062',
                     'b9072','b9092','b9102','b9082','ea_loss']
outflow_variables_cat = list(set(outflow_variables) - set(numeric_columns))
print('Outflow categorical variables:')
print(outflow_variables_cat)


tax_variables = ['taxes','b3102','b3112','b3122','b3132']
tax_variables_cat = list(set(tax_variables) - set(numeric_columns))
print('Tax categorical variables:')
print(tax_variables_cat)


family_variables = ['fsize','z2011_h_sex','z2021_h_age','z2031_h_ms',
                    'z2041_h_educ','z2051_h_has_job','z2061_h_occup',
                    'z2071_h_kb','z2081_h_cw','z2091_hhld_type']
family_variables_cat = list(set(family_variables) - set(numeric_columns))
family_variables_num = list(set(family_variables) - set(family_variables_cat))
print('Family categorical variables:')
print(family_variables_cat)


house_variables = ['b4011_bldg_type','b4021_roof','b4031_walls','b4041_tenure',
                   'b4043_house_rent','b4053_lot_rent',
                   'b4081_hse_altertn','b5012_oth_house','b5021_toilet',
                   'b5031_electric','b5041_water','b5051_w_radio','b5052_n_radio',
                   'b5061_w_tv','b5062_n_tv','b5071_vtr','b5072_n_vtr','b5081_w_stereo',
                   'b5091_w_ref','b5092_n_ref','b5101_w_wash','b5102_n_wash',
                   'b5111_w_aircon','b5121_w_salaset','b5122_n_salaset',
                   'b5131_w_dining','b5132_n_dining','b5141_w_car','b5142_n_car',
                   'b5151_w_phone','b5152_n_phone','b5161_w_pc','b5162_n_pc','b5171_w_oven',
                   'b5172_n_oven','b5181_w_motor','b5182_n_motor','w_no_hh']
house_variables_cat = list(set(house_variables) - set(numeric_columns))
house_variables_num = list(set(house_variables) - set(house_variables_cat))
print('House categorical variables:')
print(house_variables_cat)


geog_variables = ['w_prv','w_urb2','w_str2']
geog_variables_cat = list(set(geog_variables) - set(numeric_columns))
geog_variables_num = list(set(geog_variables) - set(geog_variables_cat))
print('Geographic categorical variables:')
print(geog_variables_cat)


other_variables = ['rfact','majsr','minsr','agind']
other_variables_cat = list(set(other_variables) - set(numeric_columns))
other_variables_num = list(set(other_variables) - set(other_variables_cat))
```

```
print('Other categorical variables:')
print(other_variables_cat)
```

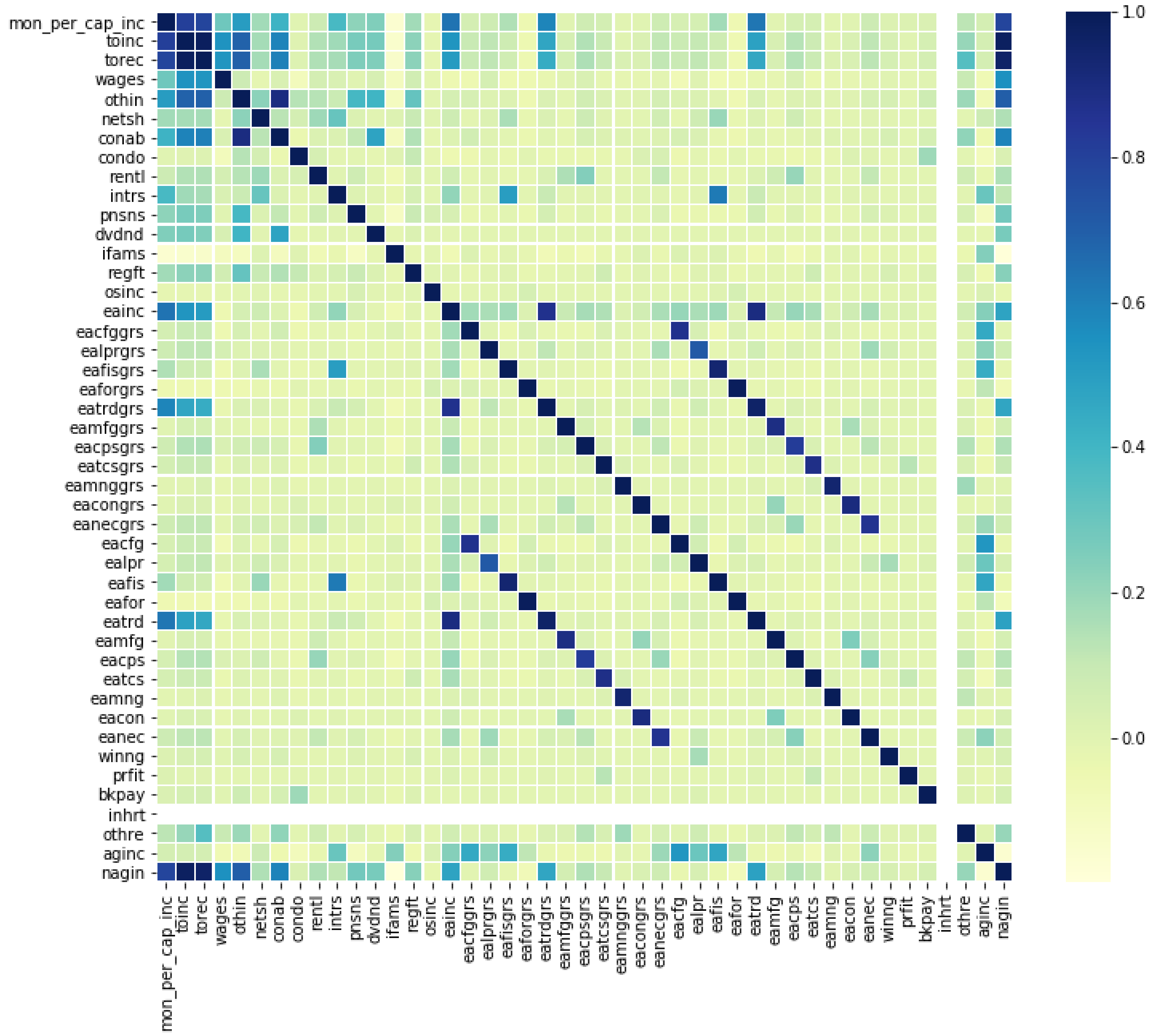
```
Inflow categorical variables:
[]
Expense categorical variables:
[]
Outflow categorical variables:
[]
Tax categorical variables:
[]
Family categorical variables:
['z2041_h_educ', 'z2081_h_cw', 'z2091_hhld_type', 'z2051_h_has_job', 'z2031_h_ms', 'z2011_h_sex']
House categorical variables:
['b5031_electric', 'b5081_w_stereo', 'b4031_walls', 'b5051_w_radio', 'b4021_roof', 'b5101_w_wash', 'b5161_w_pc', 'b5
Geographic categorical variables:
['w_urb2']
Other categorical variables:
['agind', 'minsr', 'majsr']
```

2. Check the correlation of variables

We trim down further the feature variables and include only those with low and moderate correlation with *toinc* and monthly per capita income.

```
corr_matrix = data_df[['mon_per_cap_inc','toinc'] + inflow_variables].corr()
f, ax = plt.subplots(figsize =(12, 10))
sns.heatmap(corr_matrix, ax = ax, cmap ="YlGnBu", linewidths = 0.1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fcee0a06898>
```



Notes

1. *toinc* is highly correlated with *torec* and *nagin* in the inflow group.
2. *toinc* is highly correlated with *b9072* in the outflow group.

```
main_variables = list(set(main_variables) - set(['torec','z2101_tot_mem']))
inflow_variables = list(set(inflow_variables) - set(['torec','nagin']))
outflow_variables = list(set(outflow_variables) - set(['b9072']))

numeric_variables_copy = list(set(numeric_columns) - set(['torec','z2101_tot_mem','nagin', 'b9072', 'mon_per_cap_inc']))
```

3. Modeling

Initially, backward selection was applied to all the identified variables to eliminate statistically insignificant feature variables. But because of programming (python) limitations, it was not successful. Instead we opt to select only represenative variables for each group. For logistic modeling to proceed, we only include *taxes*, *fsize*, *othin*, *totex*, and *wages*.

```
nrows = len(data_df)
train_index, test_index = train_test_split(list(range(nrows)), test_size=0.20, random_state=SEED_NUMBER)

print('Feature variables to consider: ', main_variables)
X = data_df[main_variables[:3]].iloc[train_index,]
X['intercept'] = 1
y = data_df['poverty'][train_index]

logit_model = sm.Logit(y, X)
result = logit_model.fit()
print(result.summary2())
```

```
Feature variables to consider:  ['taxes', 'wages', 'othin', 'fsize', 'totex']
Optimization terminated successfully.
         Current function value: 0.449541
         Iterations 10
                          Results: Logit
=================================================================
Model:              Logit            Pseudo R-squared: 0.222
Dependent Variable: poverty          AIC:              1871.7984
Date:               2020-03-13 09:06 BIC:              1894.3454
No. Observations:   2073             Log-Likelihood:   -931.90
Df Model:           3                LL-Null:          -1197.3
Df Residuals:       2069             LLR p-value:      1.0275e-114
Converged:          1.0000           Scale:            1.0000
No. Iterations:     10.0000
------------------------------------------------------------------
               Coef.   Std.Err.     z      P>|z|    [0.025   0.975]
------------------------------------------------------------------
taxes         -0.0004    0.0001   -4.0683  0.0000  -0.0006  -0.0002
wages         -0.0000    0.0000   -6.7110  0.0000  -0.0000  -0.0000
othin         -0.0000    0.0000  -12.1964  0.0000  -0.0000  -0.0000
intercept      0.8743    0.1191    7.3391  0.0000   0.6408   1.1077
=================================================================
```

```
x_new = data_df[main_variables[:3]].iloc[test_index,]
x_new['intercept'] = 1
y_pred_log = (result.predict(x_new) >= 0.5)*1
y_target = data_df['poverty'][test_index]

print(metrics.classification_report(y_target, y_pred_log))

print('True poverty rate:\t',  sum(y_target)/len(y_target))
print('Predicted poverty rate:\t',  sum(y_pred_log)/len(y_pred_log))
```

```
              precision    recall  f1-score   support

           0       0.79      0.89      0.84       390
           1       0.45      0.27      0.34       129

    accuracy                           0.74       519
   macro avg       0.62      0.58      0.59       519
weighted avg       0.70      0.74      0.71       519

True poverty rate:       0.24855491329479767
Predicted poverty rate:  0.15028901734104047
```

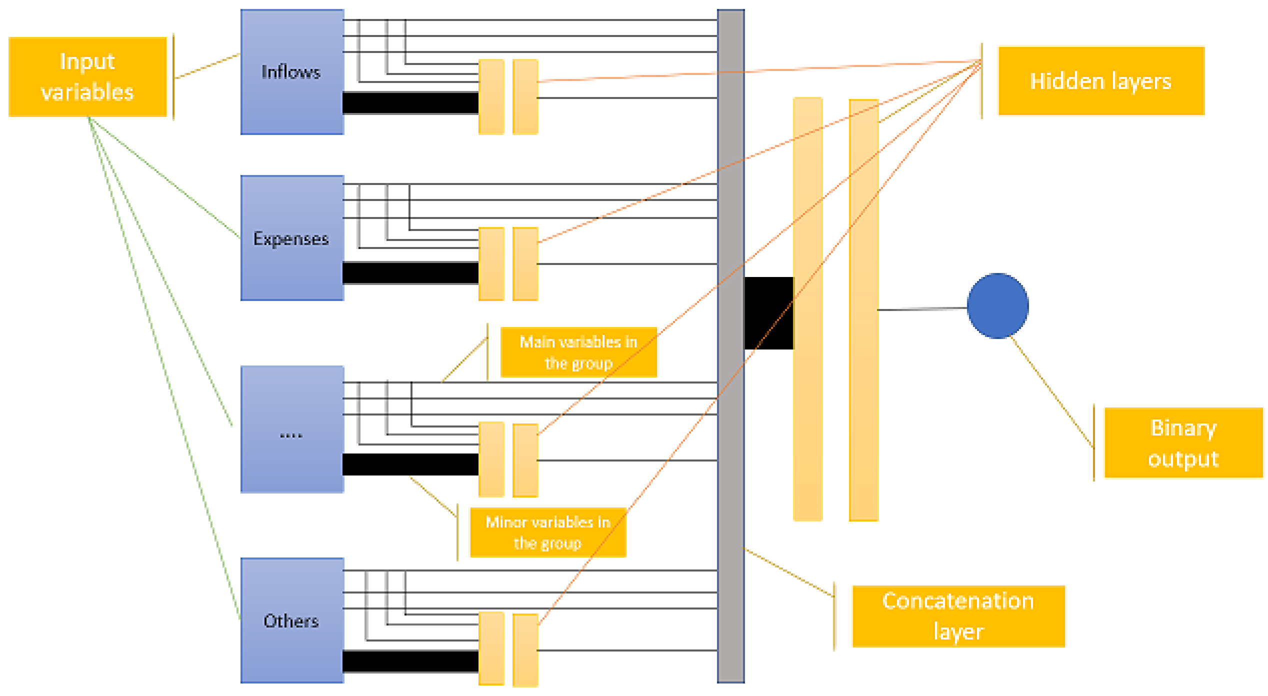
Among all the possible combinations we tried, the combination of *taxes*, *fsize*, and *othin* had significance. The accuracy is 72%. We choose this as an benchmark alternative model.

B. Deep Neural Network (extension)

For this extended DNN model, the following architecture is used. Here, the hidden sublayers between the input layer and cincatenation layer capture the deeper structure in each group. They maybe considered as adjustment factor that considers other minor variables within the group. The output of these *adjustment factors* will join the main variables *wages*, *othin*, *totex*, *taxes*, *fsize*, and *z2101_tot_mem*.

1. Architecture

```
from IPython.display import Image
Image(GOOGLE_DRIVE_STAT219_PATH + "/Alternate DNN Architecture.PNG")
```




```
categorical_variables = family_variables_cat + house_variables_cat + geog_variables_cat + other_variables_cat


def encode_categorical(input_matrix):
  enc = OneHotEncoder()
  enc.fit(input_matrix)
  return enc.transform(input_matrix).toarray()

def scale_numeric(input_matrix):
  scaler = MinMaxScaler(feature_range=(0, 1))
  scaler = scaler.fit(input_matrix)
  return scaler.transform(input_matrix)

  # load the dataset
def load_dataset(data):

  # split into input (X) and output (y) variables
  # X0: Main variables
  # X1: Inflows     X2: Expenses     X3: Outflows     X4: Taxes
  # X5: Family      X6: House        X7: Geography    X8: Others

  X0_Num = data[main_variables].values

  X1_Num = data[inflow_variables].values
  X2_Num = data[expense_variables].values
  X3_Num = data[outflow_variables].values
  X4_Num = data[tax_variables].values

  X5_Num = data[family_variables_num].values
  X5_Cat = data[family_variables_cat].values

  X6_Num = data[house_variables_num].values
  X6_Cat = data[house_variables_cat].values

  X7_Num = data[geog_variables_num].values
  X7_Cat = data[geog_variables_cat].values

  X8_Num = data[other_variables_num].values
  X8_Cat = data[other_variables_cat].values

  # Normalize numerical inputs
  X0 = scale_numeric(X0_Num)
  X1 = scale_numeric(X1_Num)
  X2 = scale_numeric(X2_Num)
  X3 = scale_numeric(X3_Num)
  X4 = scale_numeric(X4_Num)
  X5_Num = scale_numeric(X5_Num)
  X6_Num = scale_numeric(X6_Num)
  X7_Num = scale_numeric(X7_Num)
  X8_Num = scale_numeric(X8_Num)
```

```
  # format and encode all categorical inputs as string
  X5_Cat = encode_categorical(X5_Cat.astype(str))
  X6_Cat = encode_categorical(X6_Cat.astype(str))
  X7_Cat = encode_categorical(X7_Cat.astype(str))
  X8_Cat = encode_categorical(X8_Cat.astype(str))

  # combine categorical and numerical fields
  X5 = np.hstack((X5_Num, X5_Cat))
  X6 = np.hstack((X6_Num, X6_Cat))
  X7 = np.hstack((X7_Num, X7_Cat))
  X8 = np.hstack((X8_Num, X8_Cat))

  # reshape target to be a 2d array
  y_reg = data[output_variables[0]].values
  y_reg = scale_numeric(y_reg.reshape((len(y_reg), 1)))
  y_bin = data[output_variables[1]].values
  y_bin = y_bin.reshape((len(y_bin), 1))
  y = np.hstack((y_reg, y_bin))

  return X0, X1, X2, X3, X4, X5, X6, X7, X8, y


X_main, X_inflow, X_expense, X_outflow, X_tax, X_family, X_house, X_geog, X_others, y = load_dataset(data_df)


nrows = len(X_main)
train_index, test_index = train_test_split(list(range(nrows)), test_size=0.20, random_state=SEED_NUMBER)

X_main_train, X_main_test = X_main[train_index, :], X_main[test_index, :]
X_inflow_train, X_inflow_test = X_inflow[train_index, :], X_inflow[test_index, :]
X_expense_train, X_expense_test = X_expense[train_index, :], X_expense[test_index, :]
X_outflow_train, X_outflow_test = X_outflow[train_index, :], X_outflow[test_index, :]
X_tax_train, X_tax_test = X_tax[train_index, :], X_tax[test_index, :]
X_family_train, X_family_test = X_family[train_index, :], X_family[test_index, :]
X_house_train, X_house_test = X_house[train_index, :], X_house[test_index, :]
X_geog_train, X_geog_test = X_geog[train_index, :], X_geog[test_index, :]
X_others_train, X_others_test = X_others[train_index, :], X_others[test_index, :]
y_train, y_test = y[train_index, :], y[test_index, :]


print('X_main_train shape: ', X_main_train.shape)
print('X_inflow_train shape: ', X_inflow_train.shape)
print('X_expense_train shape: ', X_expense_train.shape)
print('X_outflow_train shape: ', X_outflow_train.shape)
print('X_tax_train shape: ', X_tax_train.shape)
print('X_family_train shape: ', X_family_train.shape)
print('X_house_train shape: ', X_house_train.shape)
print('X_geog_train shape: ', X_geog_train.shape)
print('X_others_train shape: ', X_others_train.shape)
print('y_train shape: ', y_train.shape)
```

```
X_main_train shape:  (2073, 5)
X_inflow_train shape:  (2073, 41)
X_expense_train shape:  (2073, 154)
X_outflow_train shape:  (2073, 12)
X_tax_train shape:  (2073, 5)
X_family_train shape:  (2073, 37)
X_house_train shape:  (2073, 82)
X_geog_train shape:  (2073, 4)
X_others_train shape:  (2073, 26)
y_train shape:  (2073, 2)
```

2. Modeling

First, we try using only a single hidden later right before the output.

```
input_MAIN = keras.layers.Input(shape = [X_main_train.shape[1]], name = 'main_input')
input_INFLOW = keras.layers.Input(shape = [X_inflow_train.shape[1]], name = 'inflow_input')
input_EXPENSE = keras.layers.Input(shape = [X_expense_train.shape[1]], name = 'expense_input')
input_OUTFLOW = keras.layers.Input(shape = [X_outflow_train.shape[1]], name = 'outflow_input')
input_TAX = keras.layers.Input(shape = [X_tax_train.shape[1]], name = 'tax_input')
input_FAMILY = keras.layers.Input(shape = [X_family_train.shape[1]], name = 'family_input')
input_HOUSE = keras.layers.Input(shape = [X_house_train.shape[1]], name = 'house_input')
input_GEOG = keras.layers.Input(shape = [X_geog_train.shape[1]], name = 'geog_input')
input_OTHERS = keras.layers.Input(shape = [X_others_train.shape[1]], name = 'others_input')

# Inflows module
hidden_INFLOW_1 = keras.layers.Dense(300, activation='elu', kernel_initializer='he_normal')(input_INFLOW)
dropout_INFLOW_1 = keras.layers.Dropout(rate=0.2)(hidden_INFLOW_1)
```

```
dropout_INFLOW_1 = keras.layers.Dropout(rate=0.2)(hidden_INFLOW_1)
hidden_INFLOW_2 = keras.layers.Dense(300, activation='elu', kernel_initializer='he_normal')(dropout_INFLOW_1)
dropout_INFLOW_2 = keras.layers.Dropout(rate=0.2)(hidden_INFLOW_2)
adjustment_factor_INFLOW = keras.layers.Dense(1, activation='relu', kernel_initializer='he_normal')(dropout_INFLOW_2)

# Expenses module
hidden_EXPENSE_1 = keras.layers.Dense(300, activation='elu', kernel_initializer='he_normal')(input_EXPENSE)
dropout_EXPENSE_1 = keras.layers.Dropout(rate=0.2)(hidden_EXPENSE_1)
hidden_EXPENSE_2 = keras.layers.Dense(300, activation='elu', kernel_initializer='he_normal')(dropout_EXPENSE_1)
dropout_EXPENSE_2 = keras.layers.Dropout(rate=0.2)(hidden_EXPENSE_2)
adjustment_factor_EXPENSE = keras.layers.Dense(1, activation='relu', kernel_initializer='he_normal')(dropout_EXPENSE_2)

# Outflows module
hidden_OUTFLOW_1 = keras.layers.Dense(300, activation='elu', kernel_initializer='he_normal')(input_OUTFLOW)
dropout_OUTFLOW_1 = keras.layers.Dropout(rate=0.2)(hidden_OUTFLOW_1)
hidden_OUTFLOW_2 = keras.layers.Dense(300, activation='elu', kernel_initializer='he_normal')(dropout_OUTFLOW_1)
dropout_OUTFLOW_2 = keras.layers.Dropout(rate=0.2)(hidden_OUTFLOW_2)
adjustment_factor_OUTFLOW = keras.layers.Dense(1, activation='relu', kernel_initializer='he_normal')(dropout_OUTFLOW_2)

# Taxes module
hidden_TAX_1 = keras.layers.Dense(300, activation='elu', kernel_initializer='he_normal')(input_TAX)
dropout_TAX_1 = keras.layers.Dropout(rate=0.2)(hidden_TAX_1)
hidden_TAX_2 = keras.layers.Dense(300, activation='elu', kernel_initializer='he_normal')(dropout_TAX_1)
dropout_TAX_2 = keras.layers.Dropout(rate=0.2)(hidden_TAX_2)
adjustment_factor_TAX = keras.layers.Dense(1, activation='relu', kernel_initializer='he_normal')(dropout_TAX_2)

# Family module
hidden_FAMILY_1 = keras.layers.Dense(300, activation='elu', kernel_initializer='he_normal')(input_FAMILY)
dropout_FAMILY_1 = keras.layers.Dropout(rate=0.2)(hidden_FAMILY_1)
hidden_FAMILY_2 = keras.layers.Dense(300, activation='elu', kernel_initializer='he_normal')(dropout_FAMILY_1)
dropout_FAMILY_2 = keras.layers.Dropout(rate=0.2)(hidden_FAMILY_2)
adjustment_factor_FAMILY = keras.layers.Dense(1, activation='relu', kernel_initializer='he_normal')(dropout_FAMILY_2)

# Household module
hidden_HOUSE_1 = keras.layers.Dense(300, activation='elu', kernel_initializer='he_normal')(input_HOUSE)
dropout_HOUSE_1 = keras.layers.Dropout(rate=0.2)(hidden_HOUSE_1)
hidden_HOUSE_2 = keras.layers.Dense(300, activation='elu', kernel_initializer='he_normal')(dropout_HOUSE_1)
dropout_HOUSE_2 = keras.layers.Dropout(rate=0.2)(hidden_HOUSE_2)
adjustment_factor_HOUSE = keras.layers.Dense(1, activation='relu', kernel_initializer='he_normal')(dropout_HOUSE_2)

# Geographic module
hidden_GEO_1 = keras.layers.Dense(300, activation='elu', kernel_initializer='he_normal')(input_GEOG)
dropout_GEOG_1 = keras.layers.Dropout(rate=0.2)(hidden_GEO_1)
hidden_GEOG_2 = keras.layers.Dense(300, activation='elu', kernel_initializer='he_normal')(dropout_GEOG_1)
dropout_GEOG_2 = keras.layers.Dropout(rate=0.2)(hidden_GEOG_2)
adjustment_factor_GEOG = keras.layers.Dense(1, activation='relu', kernel_initializer='he_normal')(dropout_GEOG_2)

# Others module
hidden_OTHERS_1 = keras.layers.Dense(300, activation='elu', kernel_initializer='he_normal')(input_OTHERS)
dropout_OTHERS_1 = keras.layers.Dropout(rate=0.2)(hidden_OTHERS_1)
hidden_OTHERS_2 = keras.layers.Dense(300, activation='elu', kernel_initializer='he_normal')(dropout_OTHERS_1)
dropout_OTHERS_2 = keras.layers.Dropout(rate=0.2)(hidden_OTHERS_2)
adjustment_factor_OTHERS = keras.layers.Dense(1, activation='relu', kernel_initializer='he_normal')(dropout_OTHERS_2)

# Combined effects and final layer
concat = keras.layers.concatenate([input_MAIN,
                                   adjustment_factor_INFLOW,
                                   adjustment_factor_EXPENSE,
                                   adjustment_factor_OUTFLOW,
                                   adjustment_factor_TAX,
                                   adjustment_factor_FAMILY,
                                   adjustment_factor_HOUSE,
                                   adjustment_factor_GEOG,
                                   adjustment_factor_OTHERS])
hidden_LAST_1 = keras.layers.Dense(300, activation='elu', kernel_initializer='he_normal')(concat)
output_bin = keras.layers.Dense(1, activation='sigmoid', name='output_bin')(hidden_LAST_1)
output_reg = keras.layers.Dense(1, activation='softplus', name='output_reg')(hidden_LAST_1)
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:541: The name tf.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:66: The name tf.g
```

```
model_7 = None

model_7 = keras.Model(inputs=[input_MAIN, input_INFLOW, input_EXPENSE, input_OUTFLOW,
                            input_TAX, input_FAMILY, input_HOUSE, input_GEOG, input_OTHERS],
                    outputs=[output_reg, output_bin])

# callbacks
output_name = GOOGLE_DRIVE_STAT219_PATH + '/models/deep_model_3_'
valid_x = [X_main_test, X_inflow_test, X_expense_test, X_outflow_test,
           X_tax_test, X_family_test, X_house_test, X_geog_test, X_others_test]
valid_y = [y_test[:,0],y_test[:,1]]

two_checkpoint_cb = callbacks.ModelCheckpoint(output_name + 'model_{epoch:03d}_{output_bin_acc:03f}_{val_output_bin_acc:0
                          save_best_only=True)
early_stoppping_cb = callbacks.EarlyStopping(patience = 10,
                                             monitor='val_output_bin_acc',
                                             restore_best_weights=True)

model_7.compile(loss=['mse','binary_crossentropy'],
              optimizer='Nadam',
              metrics=[keras.metrics.mean_absolute_error, 'accuracy'])

history_7 = model_7.fit(x=[X_main_train, X_inflow_train, X_expense_train, X_outflow_train,
             X_tax_train, X_family_train, X_house_train, X_geog_train, X_others_train],
          y=[y_train[:,0],y_train[:,1]],
          epochs=30,
          validation_data=(valid_x, valid_y),
          callbacks=[early_stoppping_cb, two_checkpoint_cb])
```

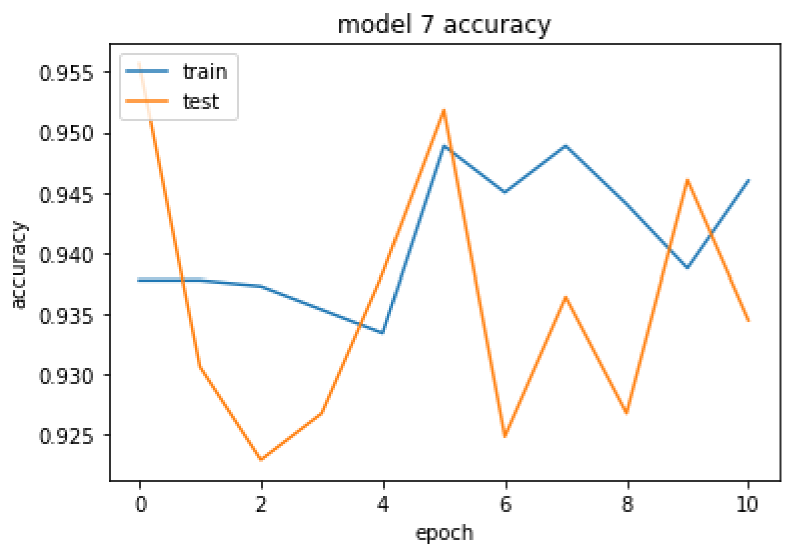
```
Train on 2073 samples, validate on 519 samples
Epoch 1/30
2073/2073 [==============================] - 5s 2ms/step - loss: 0.1616 - output_reg_loss: 0.0032 - output_bin_loss:
Epoch 2/30
2073/2073 [==============================] - 1s 516us/step - loss: 0.1492 - output_reg_loss: 0.0032 - output_bin_los
Epoch 3/30
2073/2073 [==============================] - 1s 501us/step - loss: 0.1469 - output_reg_loss: 0.0032 - output_bin_los
Epoch 4/30
2073/2073 [==============================] - 1s 506us/step - loss: 0.1530 - output_reg_loss: 0.0032 - output_bin_los
Epoch 5/30
2073/2073 [==============================] - 1s 514us/step - loss: 0.1561 - output_reg_loss: 0.0032 - output_bin_los
Epoch 6/30
2073/2073 [==============================] - 1s 511us/step - loss: 0.1399 - output_reg_loss: 0.0032 - output_bin_los
Epoch 7/30
2073/2073 [==============================] - 1s 510us/step - loss: 0.1420 - output_reg_loss: 0.0032 - output_bin_los
Epoch 8/30
2073/2073 [==============================] - 1s 503us/step - loss: 0.1336 - output_reg_loss: 0.0032 - output_bin_los
Epoch 9/30
2073/2073 [==============================] - 1s 485us/step - loss: 0.1391 - output_reg_loss: 0.0033 - output_bin_los
Epoch 10/30
2073/2073 [==============================] - 1s 513us/step - loss: 0.1458 - output_reg_loss: 0.0032 - output_bin_los
Epoch 11/30
2073/2073 [==============================] - 1s 499us/step - loss: 0.1374 - output_reg_loss: 0.0032 - output_bin_los
```

```
# Accuracy of first DNN model
y_pred = model_7.predict([X_main_test, X_inflow_test, X_expense_test, X_outflow_test,
             X_tax_test, X_family_test, X_house_test, X_geog_test, X_others_test], batch_size=64, verbose=1)

y_pred_bool = (y_pred[1] >= 0.5)*1
y_pred_bool = y_pred_bool.reshape(y_pred_bool.shape[0],)

print(metrics.classification_report(y_test[:,1], y_pred_bool))
print('True poverty rate:\t',  sum(y_test[:,1])/len(y_test[:,1]))
print('Predicted poverty rate:\t',  sum(y_pred_bool)/len(y_pred_bool))
```

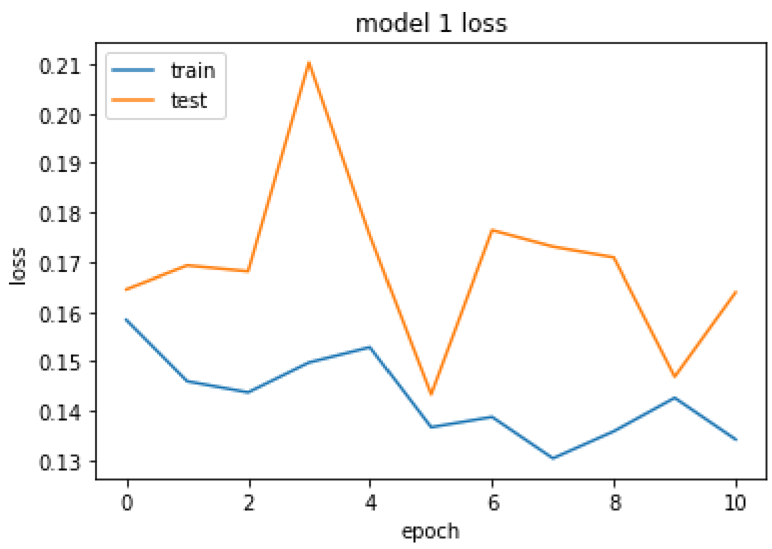
```
519/519 [==============================] - 1s 2ms/step
              precision    recall  f1-score   support

         0.0       0.99      0.95      0.97       390
         1.0       0.87      0.96      0.92       129

    accuracy                           0.96       519
   macro avg       0.93      0.96      0.94       519
weighted avg       0.96      0.96      0.96       519

True poverty rate:       0.24855491329479767
Predicted poverty rate:  0.27360308285163776
```

```
plt.plot(history_7.history['output_bin_acc'])
```

```
plt.plot(history_7.history['val_output_bin_acc'])
plt.title('model 7 accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```




```
plt.plot(history_7.history['output_bin_loss'])
plt.plot(history_7.history['val_output_bin_loss'])
plt.title('model 1 loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



For the first DNN model, we achieved a test accuracy rate of 96%. The validation accuracy rate was 94.50 %. This indicates that there was overfitting and the model is a very good choice.

```
# Combined effects and final layer for 2nd DNN model
concat = keras.layers.concatenate([input_MAIN,
                                   adjustment_factor_INFLOW,
                                   adjustment_factor_EXPENSE,
                                   adjustment_factor_OUTFLOW,
                                   adjustment_factor_TAX,
                                   adjustment_factor_FAMILY,
                                   adjustment_factor_HOUSE,
                                   adjustment_factor_GEOG,
                                   adjustment_factor_OTHERS])
hidden_LAST_1 = keras.layers.Dense(500, activation='elu', kernel_initializer='he_normal')(concat)
dropout_LAST_1 = keras.layers.Dropout(rate=0.2)(hidden_LAST_1)
hidden_LAST_2 = keras.layers.Dense(500, activation='elu', kernel_initializer='he_normal')(dropout_LAST_1)
dropout_LAST_2 = keras.layers.Dropout(rate=0.2)(hidden_LAST_2)
output_bin = keras.layers.Dense(1, activation='sigmoid', name='output_bin')(dropout_LAST_2)
output_reg = keras.layers.Dense(1, activation='softplus', name='output_reg')(dropout_LAST_2)

model_8 = keras.Model(inputs=[input_MAIN, input_INFLOW, input_EXPENSE, input_OUTFLOW,
                            input_TAX, input_FAMILY, input_HOUSE, input_GEOG, input_OTHERS],
                    outputs=[output_reg, output_bin])

# callbacks
output_name = GOOGLE_DRIVE_STAT219_PATH + '/models/deep_model_4_'
two_checkpoint_cb = callbacks.ModelCheckpoint(output_name + 'model_{epoch:03d}_{output_bin_acc:03f}_{val_output_bin_acc:0
                          save_best_only=True)
early_stoppping_cb = callbacks.EarlyStopping(patience = 10, monitor='val_output_bin_acc',
                                             restore_best_weights=True)
```

```
#monitor='val_output_bin_acc',
model_8.compile(loss=['mse','binary_crossentropy'],
              optimizer='Nadam',
              metrics=[keras.metrics.mean_absolute_error, 'accuracy'])

history_8 = model_8.fit(x=[X_main_train, X_inflow_train, X_expense_train, X_outflow_train,
              X_tax_train, X_family_train, X_house_train, X_geog_train, X_others_train],
              y=[y_train[:,0],y_train[:,1]],
              epochs=30,
              validation_data=(valid_x, valid_y),
              callbacks=[early_stoppping_cb, two_checkpoint_cb])
```

```
Train on 2073 samples, validate on 519 samples
Epoch 1/30
2073/2073 [==============================] - 6s 3ms/step - loss: 0.2542 - output_reg_loss: 0.0158 - output_bin_loss:
Epoch 2/30
2073/2073 [==============================] - 1s 549us/step - loss: 0.1738 - output_reg_loss: 0.0051 - output_bin_los
Epoch 3/30
2073/2073 [==============================] - 1s 546us/step - loss: 0.1752 - output_reg_loss: 0.0039 - output_bin_los
Epoch 4/30
2073/2073 [==============================] - 1s 540us/step - loss: 0.1509 - output_reg_loss: 0.0036 - output_bin_los
Epoch 5/30
2073/2073 [==============================] - 1s 536us/step - loss: 0.1994 - output_reg_loss: 0.0058 - output_bin_los
Epoch 6/30
2073/2073 [==============================] - 1s 540us/step - loss: 0.1512 - output_reg_loss: 0.0033 - output_bin_los
Epoch 7/30
2073/2073 [==============================] - 1s 529us/step - loss: 0.1595 - output_reg_loss: 0.0032 - output_bin_los
Epoch 8/30
2073/2073 [==============================] - 1s 545us/step - loss: 0.1442 - output_reg_loss: 0.0032 - output_bin_los
Epoch 9/30
2073/2073 [==============================] - 1s 528us/step - loss: 0.1473 - output_reg_loss: 0.0032 - output_bin_los
Epoch 10/30
2073/2073 [==============================] - 1s 549us/step - loss: 0.1532 - output_reg_loss: 0.0031 - output_bin_los
Epoch 11/30
2073/2073 [==============================] - 1s 545us/step - loss: 0.1422 - output_reg_loss: 0.0032 - output_bin_los
Epoch 12/30
2073/2073 [==============================] - 1s 544us/step - loss: 0.1363 - output_reg_loss: 0.0032 - output_bin_los
Epoch 13/30
2073/2073 [==============================] - 1s 524us/step - loss: 0.1489 - output_reg_loss: 0.0031 - output_bin_los
Epoch 14/30
2073/2073 [==============================] - 1s 540us/step - loss: 0.1438 - output_reg_loss: 0.0033 - output_bin_los
Epoch 15/30
2073/2073 [==============================] - 1s 533us/step - loss: 0.1352 - output_reg_loss: 0.0032 - output_bin_los
Epoch 16/30
2073/2073 [==============================] - 1s 535us/step - loss: 0.1314 - output_reg_loss: 0.0032 - output_bin_los
Epoch 17/30
2073/2073 [==============================] - 1s 538us/step - loss: 0.1405 - output_reg_loss: 0.0031 - output_bin_los
Epoch 18/30
2073/2073 [==============================] - 1s 529us/step - loss: 0.1338 - output_reg_loss: 0.0030 - output_bin_los
```

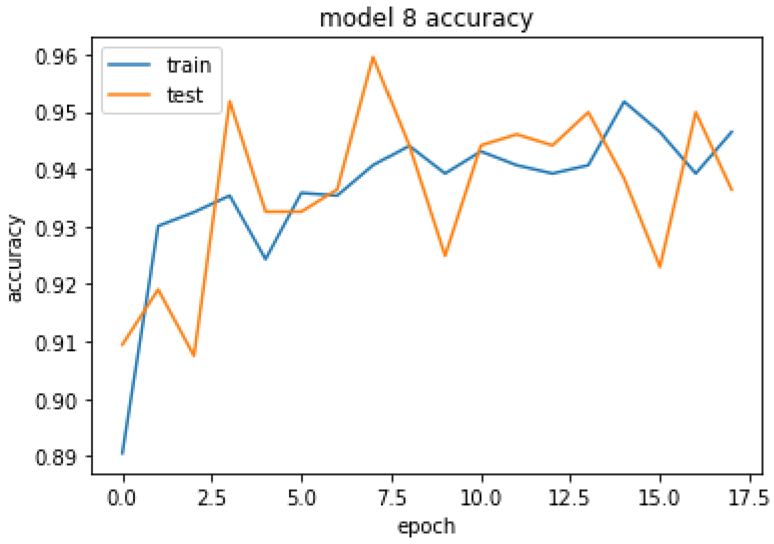
```
# Accuracy for 2nd DNN model
y_pred = model_8.predict([X_main_test, X_inflow_test, X_expense_test, X_outflow_test,
             X_tax_test, X_family_test, X_house_test, X_geog_test, X_others_test], batch_size=64, verbose=1)

y_pred_bool = (y_pred[1] >= 0.5)*1
y_pred_bool = y_pred_bool.reshape(y_pred_bool.shape[0],)

print(metrics.classification_report(y_test[:,1], y_pred_bool))
print('True poverty rate:\t',  sum(y_test[:,1])/len(y_test[:,1]))
print('Predicted poverty rate:\t',  sum(y_pred_bool)/len(y_pred_bool))
```

```
519/519 [==============================] - 1s 2ms/step
              precision    recall  f1-score   support

         0.0       0.98      0.97      0.97       390
         1.0       0.90      0.94      0.92       129

    accuracy                           0.96       519
   macro avg       0.94      0.95      0.95       519
weighted avg       0.96      0.96      0.96       519

True poverty rate:       0.24855491329479767
Predicted poverty rate:  0.2581888246628131
```

For the second DNN model, we used two final hidden layers. The best test accuracy rate was 96%. It is higher than our first model. The validation accuracy was 94.07%. Hence, **we choose this model as our best DNN model**.

```
plt.plot(history_8.history['output_bin_acc'])
plt.plot(history_8.history['val_output_bin_acc'])
plt.title('model 8 accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
```

```
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```




```
plt.plot(history_8.history['output_bin_loss'])
plt.plot(history_8.history['val_output_bin_loss'])
plt.title('model 8 loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```