III. Model documentation and write-up

1.Who are you (mini-bio) and what do you do professionally?

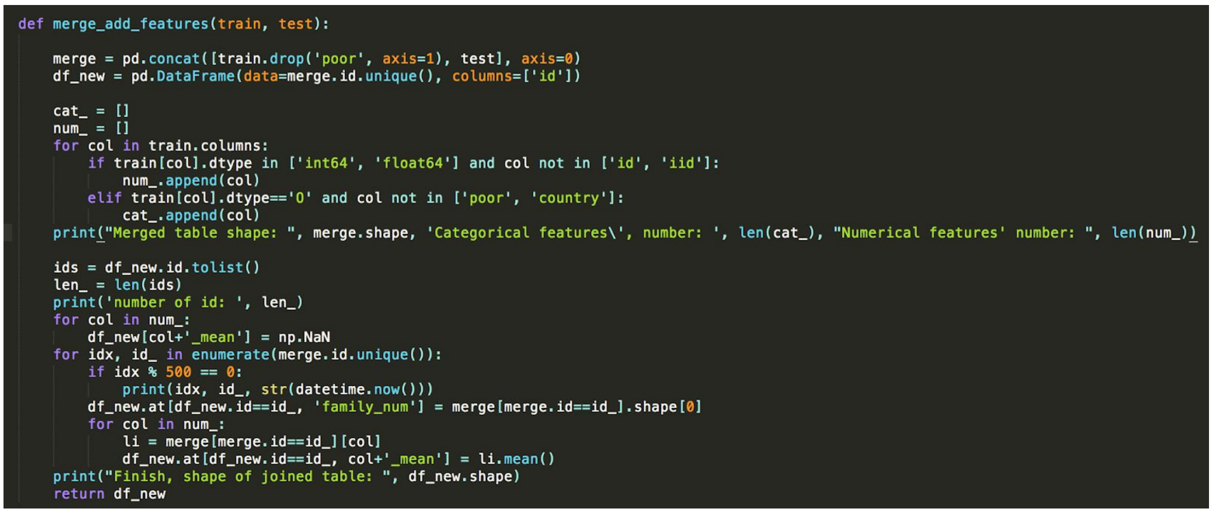
- I worked as a data scientist in a IoT startup located in Frankfurt. My daily work is to develop machine learning models for the collected data from different sensors and do analysis. I have a master degree in computer science and communication engineering. I learned machine learning all from online open courses and books.

2. High level summary of your approach: what did you do and why?

- My method is basically a combination of gradient boosted decision tree and neural network. For gradient boosted decision tree models I used Lightgbm and Xgboost open source libraries. For neural network, I used Keras to build the model. I built one model for each of country A, country B and country C. The final submission is a weighted average of 10-folds cross validation combination of Xgboost model, Lightgbm model and Neural network model for Country A and B. For country C, 10-folds cross validation combination of Xgboost and Lightgbm is used. But after this competition I found 20-folds cross validation improve local validation score a little bit, so I submit the code, where I set it to 20 folds cross validation.

3. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

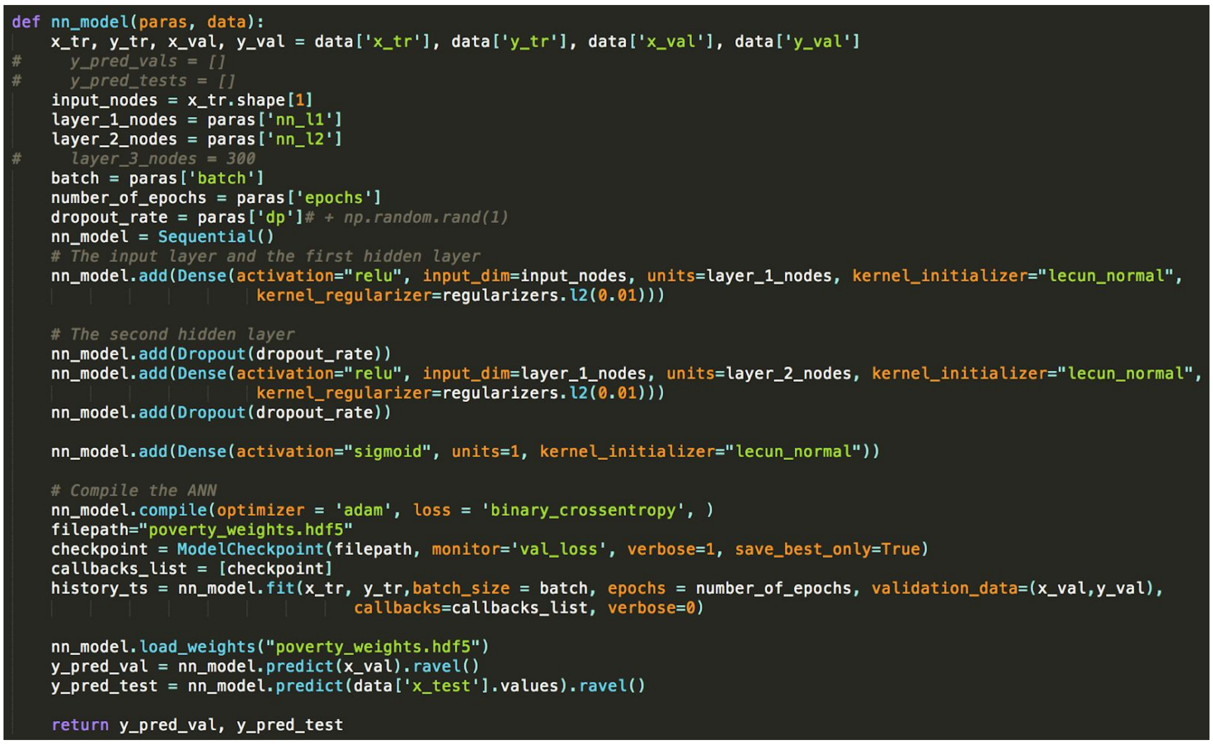
1. The first I can think of is to extract features from the *_indiv_*.csv tables. Because in the raw feature tables, two kinds of tables are provided. And one very straightforward thought would be combining these two kinds of tables into one and then build models.



```python
def merge_add_features(train, test):

    merge = pd.concat([train.drop('poor', axis=1), test], axis=0)
    df_new = pd.DataFrame(data=merge.id.unique(), columns=['id'])

    cat_ = []
    num_ = []
    for col in train.columns:
        if train[col].dtype in ['int64', 'float64'] and col not in ['id', 'iid']:
            num_.append(col)
        elif train[col].dtype=='O' and col not in ['poor', 'country']:
            cat_.append(col)
    print("Merged table shape: ", merge.shape, 'Categorical features\', number: ', len(cat_), "Numerical features' number: ", len(num_))

    ids = df_new.id.tolist()
    len_ = len(ids)
    print('number of id: ', len_)
    for col in num_:
        df_new[col+'_mean'] = np.NaN
    for idx, id_ in enumerate(merge.id.unique()):
        if idx % 500 == 0:
            print(idx, id_, str(datetime.now()))
        df_new.at[df_new.id==id_, 'family_num'] = merge[merge.id==id_].shape[0]
        for col in num_:
            li = merge[merge.id==id_][col]
            df_new.at[df_new.id==id_, col+'_mean'] = li.mean()
    print("Finish, shape of joined table: ", df_new.shape)
    return df_new
```

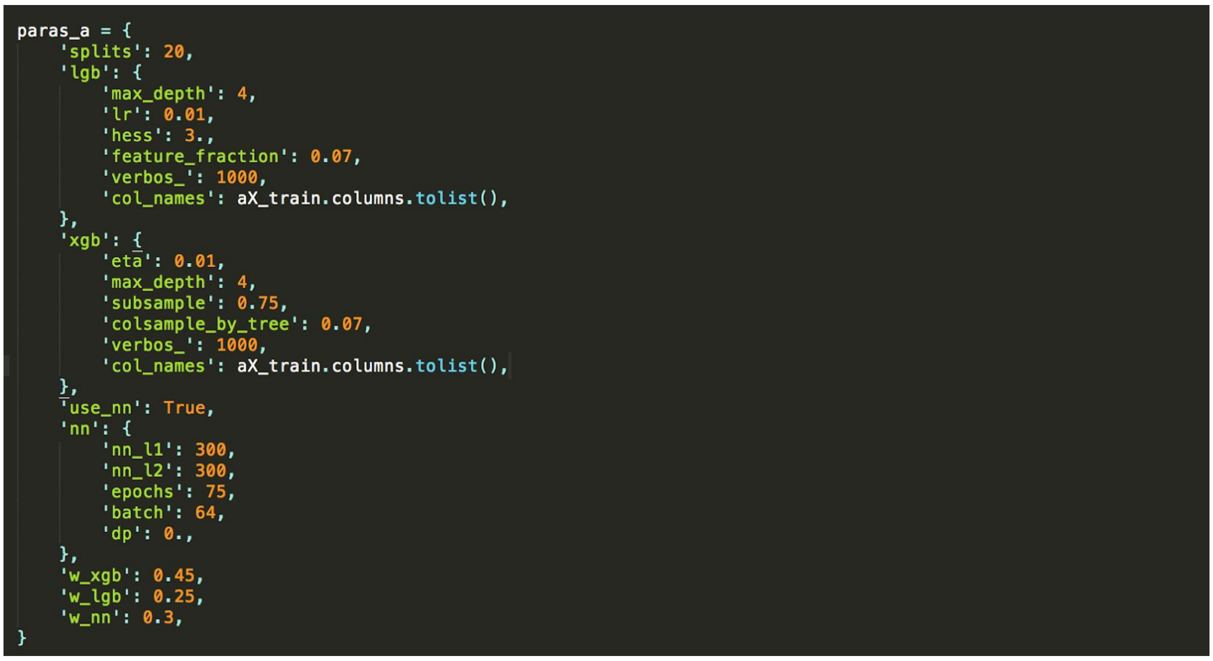
2. Another huge boost to the performance is the use of neural network. Before using it, I didn't expect too much about the neural network because of the very small dataset. Just give it a try. But it turns out for country A and country B, neural network works

great, even though the neural network I used is very simple two-layer vanilla neural network.



```python
def nn_model(paras, data):
    x_tr, y_tr, x_val, y_val = data['x_tr'], data['y_tr'], data['x_val'], data['y_val']
#     y_pred_vals = []
#     y_pred_tests = []
    input_nodes = x_tr.shape[1]
    layer_1_nodes = paras['nn_l1']
    layer_2_nodes = paras['nn_l2']
#     layer_3_nodes = 300
    batch = paras['batch']
    number_of_epochs = paras['epochs']
    dropout_rate = paras['dp']# + np.random.rand(1)
    nn_model = Sequential()
    # The input layer and the first hidden layer
    nn_model.add(Dense(activation="relu", input_dim=input_nodes, units=layer_1_nodes, kernel_initializer="lecun_normal",
                       kernel_regularizer=regularizers.l2(0.01)))

    # The second hidden layer
    nn_model.add(Dropout(dropout_rate))
    nn_model.add(Dense(activation="relu", input_dim=layer_1_nodes, units=layer_2_nodes, kernel_initializer="lecun_normal",
                       kernel_regularizer=regularizers.l2(0.01)))
    nn_model.add(Dropout(dropout_rate))

    nn_model.add(Dense(activation="sigmoid", units=1, kernel_initializer="lecun_normal"))

    # Compile the ANN
    nn_model.compile(optimizer = 'adam', loss = 'binary_crossentropy', )
    filepath="poverty_weights.hdf5"
    checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True)
    callbacks_list = [checkpoint]
    history_ts = nn_model.fit(x_tr, y_tr,batch_size = batch, epochs = number_of_epochs, validation_data=(x_val,y_val),
                              callbacks=callbacks_list, verbose=0)

    nn_model.load_weights("poverty_weights.hdf5")
    y_pred_val = nn_model.predict(x_val).ravel()
    y_pred_test = nn_model.predict(data['x_test'].values).ravel()

    return y_pred_val, y_pred_test
```

3. The final trick, well I don't know if it's a trick, is to summarize all the parameters I used in all those models into a dictionary, which makes it very easy to tune the parameters to select the best parameter set including the weights for different models. Because I joined this competition pretty late, which is 6 days before the end of the competition. This trick really helps me save a lot of time and try a lot of different technologies I want to try.



```python
paras_a = {
    'splits': 20,
    'lgb': {
        'max_depth': 4,
        'lr': 0.01,
        'hess': 3.,
        'feature_fraction': 0.07,
        'verbos_': 1000,
        'col_names': aX_train.columns.tolist(),
    },
    'xgb': {
        'eta': 0.01,
        'max_depth': 4,
        'subsample': 0.75,
        'colsample_by_tree': 0.07,
        'verbos_': 1000,
        'col_names': aX_train.columns.tolist(),
    },
    'use_nn': True,
    'nn': {
        'nn_l1': 300,
        'nn_l2': 300,
        'epochs': 75,
        'batch': 64,
        'dp': 0.,
    },
    'w_xgb': 0.45,
    'w_lgb': 0.25,
    'w_nn': 0.3,
}
```

4. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

- Actually because I joined this competition too late, I only tried very limited technologies. I spent most of time focusing on country A and then used the same tricks for country B and country C. I still don't know why neural network model didn't work well for country C. I have tested all categorical features in individual tables didn't improve the score for the country A model, but didn't test if there is any such features help the models for country B and country C. And also I didn't do too much feature selection work. In *_hhold_*.csv tables there are too many features for such small dataset. It seems there are some potentials for improving the performance by dropping some features. Finally I always want to apply stacking technology in one competition, because I hear and see too many examples, where stacking helps the performance a lot. But didn't make it this time.

5. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?
   - No

6. How did you evaluate performance of the model other than the provided metric, if at all?
   - No, just used the provided evaluation performance metrics

7. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?
   - The only thing I can think of is because I didn't set the random seed for the neural network model, so the result is not deterministic. Every time you run the code, the result and score is not totally the same, but the scores should not have a huge difference. Actually after the competition, I fine-tuned my model a little bit, it gave me much better local cross validation score than the best submission's local validation score (from 0.1609 to 0.1584).

8. Do you have any useful charts, graphs, or visualizations from the process?
   - No

9. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?
   - Definitely I will try stacking techniques. Talking about the features, because all the provided features are hash coded, so we don't know what each feature means, which limits the potential to do feature engineering. Not sure if it's necessary, but if the detail information about each feature is given, it will have much more fun.