Model documentation and write-up

1. Who are you (mini-bio) and what do you do professionally?
   I have been working in IT for many years. Started as a developer in C ++, then Java and Scala. And now I am the head of the expert systems department. I participate in the development as an architect and an expert.
   Data analysis is my hobby, like participating in various competitions.
   https://www.linkedin.com/in/taras-baranyuk-682a2b48/?locale=en_US

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

   Since there are many categorical features in the data and a small size of samples, I decided to use tree-based algorithms (CatBoost, XGBoost and LightGBM). In the final submission, I used an ensemble of the results of each algorithm. The smallest correlation was between CatBoost vs XGBoost and CatBoost vs LightGBM. So, I chose the weights 0.4 for XGBoost and CatBoost, and 0.2 for LightGBM.
   Since the data for countries are not balanced, this fact had to be taken into account in the calculations. Cross-validation showed that for countries A and B it is better to use weights in the algorithms, and for country B to make up-sampling.
   Since the features was too much, the generation of new ones had to be treated very carefully and only those that gave a significant increase in cross-validation(the number of unique categories for households, the number of residents, the number of positive and negative values). In addition, cross-validation showed that the data of their individual set for country A did not reduce the error.
   After the generation of the feature, I filtered using the feature_importance parameter for each of the algorithms separately, and this significantly reduced the number of features without loss of quality.

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 function helps to effectively reduce the number of features without losing the quality of the algorithm.

```
def find_exclude(self, n_splits=5):
        if not self.model_dict or not self.data_dict:
            print('Stoped: no models or data')
            return None

        for c in self.countries:
            self.model_dict[c].load_data(data=self.data_dict[c],
                                         balance=self.balances[c])
            exclude_list = []
            finish = False
            logloss_dict = {}
            while not finish:
                self.model_dict[c].set_exclude_list(exclude_list)
                self.model_dict[c].train()
                exclude_list_prev = exclude_list.copy()
                columns = [x for x in self.model_dict[c].get_train().columns
                           if x not in exclude_list_prev]
                exclude_list = [x for (x, y) in zip(
```

```
                    columns, self.model_dict[c].get_feature_importances()
                ) if y == 0]
            if not exclude_list:
                finish = True
        exclude_list = exclude_list_prev + exclude_list

        logloss_iter = []
        splits = self.model_dict[c].data.get_train_valid(
            n_splits=n_splits, balance=self.balances[c])

        for i in range(0, n_splits):
            self.model_dict[c].set_random_seed(i)
            train, valid = splits[i]
            self.model_dict[c].set_exclude_list(exclude_list)
            self.model_dict[c].train(train[0], train[1])
            pred = self.model_dict[c].predict(valid[0])
            logloss_iter.append(log_loss(valid[1].astype(int),
                                         pred['poor']))
        logloss = np.mean(logloss_iter)
        logloss_dict[logloss] = exclude_list
        print('loglos: {0} exclude length: {1}'.format(
            logloss, len(exclude_list)))
    self.exclude_dict[c] = logloss_dict[np.min(
        list(logloss_dict.keys()))]
    print('Country: {0} exclude length: {1}'.format(
        c, len(self.exclude_dict.get(c))))

return logloss_dict
```

2. Due to data imbalance, the question of data partitioning for cross-validation is very important.

```
def split_data(self,
               size=0.8,
               n_splits=1,
               random_state=1,
               balance=False,
               df=None):
    """
    Returns data partitions.

    Args:
        size: float, partition ratio, optional (default=0.8)
        n_splits: int, number of partitions, optional (default=1)
        random_state: int, RandomState instance, optional (default=1)
        balance: bool, resample data, optional (default=False)
        df: DataFrame, data for split, optional (default=None)

    Returns:
        List of splits.
    """

    if not isinstance(df, pd.DataFrame):
        train = self.country_df_train
    else:
        train = df
    sss = StratifiedShuffleSplit(n_splits=n_splits,
                                 test_size=1-size,
                                 random_state=random_state)
    splits = []
    for train_index, validate_index in sss.split(train, train.poor):
        df_train = train.iloc[train_index]
        if balance:
            df_train = self.resample(df_train)
        splits.append((df_train, train.iloc[validate_index]))
    return splits
```

3. The generation of these features greatly improved the prediction for countries A and B.

```
def count_iid(self, df):
        """
        Get a dataframe with a count of individuals for households.

        Args:
            df: DataFrame, dataframe with an individual level data

        Returns:
            A dataframe with a count of individuals for households.
        """
        s = df.index.get_level_values(0).value_counts()
        return s.reindex(index=self._get_id_list(df)).to_frame('iid_cnt')

    def count_neg_poz(self, df):
        """
        Get a dataframe with a count of negative and positive values for
        an individual level data.

        Args:
            df: DataFrame, dataframe with an individual level data

            Returns:
            A dataframe with a count of negative and positive values for
            an individual level data.
        """
        res_df = df.select_dtypes(include=['float64', 'int64', 'int8'])
        res_df = res_df.groupby(level=0).apply(lambda c: c.apply(
            lambda x: pd.Series(
                [(x < 0).sum(), (x >= 0).sum()])).unstack())
        res_df.columns = ['{0}_{1}'.format(i[0], i[1])
                          for i in res_df.columns]
        print('count_neg_poz size df: ', res_df.shape)
        return res_df.reindex(index=self._get_id_list(df))

    def count_unique_categories(self, df, iid=True):
        """
        Get a dataframe with a count of unique values for an individual
        level data.

        Args:
            df: DataFrame, dataframe with an individual level data
            iid: bool, add columns with the ratio of the number of unique
                values to the number of individuals in households,
                optional (default=True)

        Returns:
            A dataframe with a count of unique values for an individual
            level data.
        """
        res_df = df.groupby(level=0).apply(
            lambda c: c.apply(lambda x: pd.Series([len((x).unique())])))
        res_df.index = res_df.index.droplevel(1)
            res_df.columns = [
            '{0}_{1}'.format('cat_n', i) for i in res_df.columns]
        print('count_unique_categories size df: ', res_df.shape)
        res_df = res_df.reindex(index=self._get_id_list(df))
        if iid:
            div_df = res_df.div(self.count_iid(df)['iid_cnt'], axis=0)
            div_df.columns = ['{0}_{1}'.format('div_cat_iid', i)
                              for i in res_df.columns]
            res_df = pd.concat([res_df, div_df], axis=1)
        return res_df
```

4. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?
   I tried to use linear algorithms, many different ways of generating new features.

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?
   AUC for early stopping.

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.)?
   Do not use the GPU - the quality is significantly reduced.

8. Do you have any useful charts, graphs, or visualizations from the process?
   Nothing interesting.

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?
   It is necessary to optimize hyper parameters for CatBoost and necessarily add to the ensemble the results of the work of neural networks. You can try, for each of the algorithms perform calculations with different hyper parameters and average the results.