Alternative Credit Scoring Report

I. Definition

I.I. Project Overview

The project is situated in the domains of financial risk, specifically the assessment of creditworthiness of customers applying for financial products such loans or credit cards. The basis for accessing such financial products is a credit score of the applicant. The current credit scoring models focus heavily on 1) a well-established history of (positive) interactions with financial institutions and 2) proof of steady future income as for example evidenced by permanent employment contracts.

I.II. Problem Statement

However, this leaves a series of groups with none or inadequate access to financial products such as students, gig economy workers, and migrants. My capstone project will focus on exploring the predictive power of alternative data points to correctly calculate credit scores, such as age, assets, and education. By proving adequate predictive power of those data points that could be verified during the loan application, we can make a case for using alternative credit scoring models so that the aforementioned groups can access financial products.

I.III. Metrics

According to research by Oliver Wyman, a leading management consulting company, in their paper on Alternative Data and the Unbanked[1] the cut-off rate for obtaining loans currently is at around 4% bad-rate. If we think that a 'positive' is someone who is lendable and a 'negative' is someone who is non-lendable, the banks essentially accept a false positive rate of 4%. Hence we can formulate the following optimization problem: maximize the number of true positives subject to the false positive rate being below 4%. This maximization problem, however, is only relevant if we can discern a significant difference in predictive performance between the models – hence we will validate that first by considering a variety of evaluation metrics such as f1 score, recall, precision and accuracy.

NB: I will add screenshots of relevant notebook sections to help evaluators track how the project report fits together with the python code.

[1] Source: https://www.oliverwyman.com/content/dam/oliver-wyman/v2/publications/2017/may/Alternative_Data_And_The_Unbanked.pdf

II. Analysis

II.I. Data Exploration, Exploratory Visualization, and Data Preprocessing

My analysis is based on a dataset from a Chinese financial institutions, which has been posted on Kaggle[2]. The dataset consists of two parts which are linked by a unique customer ID: first, data on 439,000 customers along 18 variables and second, data on the credit repayment history of those customers. I've also uploaded a dictionary which explains the variables and is copied from the additional information section on the Kaggle page.

Some data exploration help us get a better feeling for how to best approach the problem. First, the credit data does not have a straightforward feature of 'default'. Instead it tracks the severity of late payments per customers over time based on a number of 1 to 5. A look at the dictionary reveals that 5 corresponds to a write-off – essentially the payment is so late that the bank has no hope of recouping their money again. Hence, we use the 5 to engineer the 'defaulted' feature to train a model that essentially predicts credit worthiness based on whether an applicant is likely to default on a loan.

```
In [4]: #Review what credit record status corresponds to a default -> a '5' corresponds to default
        df_d.loc[df_d['Feature name'] == 'STATUS']['Remarks'].item()

Out[4]: '0: 1-29 days past due 1: 30-59 days past due 2: 60-89 days overdue 3: 90-119 days overdue 4: 120-149 days overdue 5:
        Overdue or bad debts write-offs for more than 150 days C: paid off that month X: No loan for the month'
```

```
#Now we know that a '5' corresponds to a severely overdue credit or a bed debt write-off.
#We can now engineer this feature and find out how many people defaulted on their products

df_cr['Defaulted'] = [1 if x == '5' else 0 for x in df_cr['STATUS']]

df_cr_aggregated = df_cr.groupby('ID', as_index = False).sum()

default_rate = len(df_cr_aggregated.loc[df_cr_aggregated['Defaulted'] != 0]) / len(df_cr_aggregated)

print('The number of people with loan information is {}.'.format(len(df_cr_aggregated)))
print('The number of people who defaulted on their loan is {}.'.format(len(df_cr_aggregated.loc[df_cr_aggregated['Defau
print('The share of people who defaulted on their loan is {}.'.format(default_rate))

The number of people with loan information is 45985.
The number of people who defaulted on their loan is 195.
The share of people who defaulted on their loan is 0.004240513210829618.
```

In a second step and once we have engineered this feature for all applicants, we note that the number of customers who defaulted is quite small (0.4%). This is problematic, since vastly unbalanced datasets encourage algorithms to focus on the larger observation classes or sometimes just 'cheat' altogether by predicting that every observation belongs to the largest class. For example, in a dataset where 99.6% of observations are non-defaulted, a naïve model that predicts that everyone is non-defaulted would reach an accuracy of 99.6%. Hence, I will create a subset that consists 10% of defaulted customers and 90% of customers who successfully repaid their loans to simulate the real world. We select the 90% at random to ensure that they are representative of the overall observations of non-defaulted applicants. We then merge the default variable with the other applicant information.

[2] Source: https://www.kaggle.com/rikdifos/credit-card-approval-prediction

```
In [7]: #Create random sample out of non-defaulted applicants and append defaulted applicants
        df_subset = df_cr_aggregated.loc[df_cr_aggregated['Defaulted']==0].sample(9*195).append(df_cr_aggregated.loc[df_cr_aggr

        #Merge that information with applicant data
        result = pd.merge(df_subset, df_ar, on=['ID'])
        print(len(result))
        result.head(10)

        1566
```

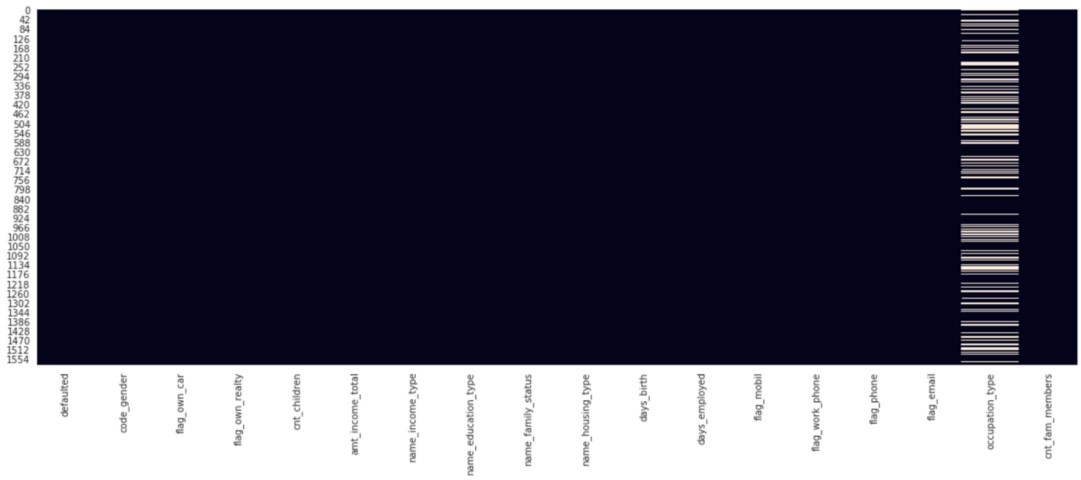
Out[7]:

	ID	MONTHS_BALANCE	Count	Defaulted	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCC
0	5053616	-28	8	0	F	N	Y	0	112500.0	
1	5054560	-3	2	0	M	N	N	0	135000.0	
2	5090451	-703	37	0	F	N	N	0	139500.0	
3	5139814	-1020	40	0	M	N	Y	0	108000.0	
4	5116091	-1326	12	0	M	N	Y	0	225000.0	
5	5088805	-77	11	0	M	Y	Y	0	225000.0	St
6	5099718	-595	35	0	M	Y	N	0	247500.0	
7	5009850	-598	15	0	M	Y	N	0	360000.0	Commercia
8	5090042	-300	24	0	M	N	N	1	225000.0	
9	5010215	-1035	37	0	F	Y	Y	1	90000.0	St

Our third step is to convert variables codified as strings into a numerical format so that our models can be trained based on them. On the one hand, this requires a simple transformation of binary variables such as 'code_gender' into 1 for male and 0 for female. For multi-class variables such as 'name_income_type' I checked that the number of possible values to ensure that none are included with too many. As a result, I had to throw out 'occupation_type', which had more than 18 possible values and over 400 observations with missing data. There were no nans in the other variables:

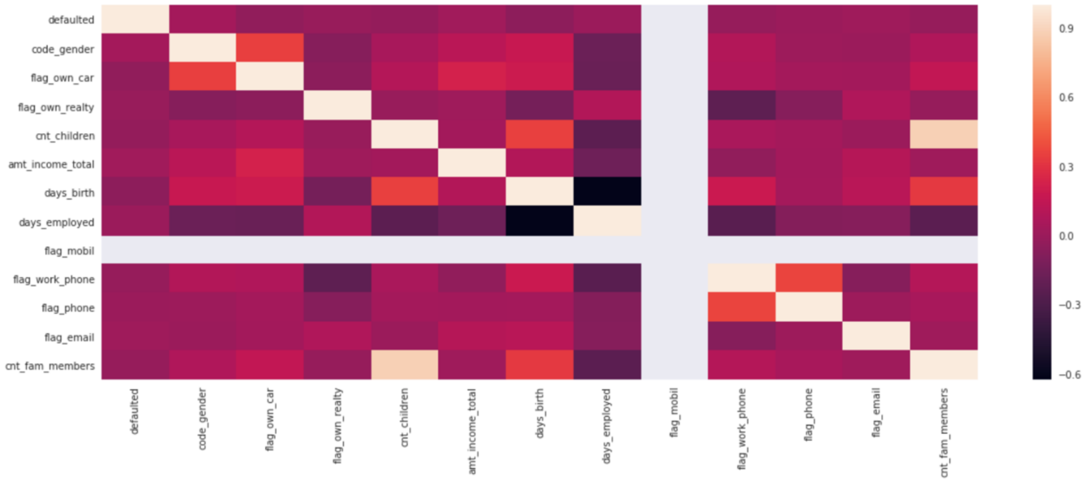
```
In [14]: sns.heatmap(result.isnull(), cbar=False)

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7f106c12e278>
```



Lastly, we should be concerned about the correlation between our variables. To better understand this, we use seaborn's heatmap functionality:

```
In [22]: sns.heatmap(result.corr())
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7f106ad72240>
```



We can see that the correlation between count children and count family members is predictably high. However, it is not so high as to be a substantial concern, so I choose to leave it in. No other substantial correlations are detected. However, interestingly the analysis shows a curious issue with the mobile ownership variable, which requires a deeper look:

```
In [25]: result['flag_mobil'].nunique()
Out[25]: 1
```

As expected, it seems that all of our applicants have a phone, which makes this variable meaningless for an algorithm trying to distinguish between them. We therefore drop this variable, too:

```
In [27]: #drop the the mobile ownership variable
         result.drop('flag_mobil', axis = 1, inplace=True)
```

II.II. Algorithms and Techniques

Now that the data is finalized, we create one dataset with only traditional data, i.e. representing the type of information usually available to traditional risk scoring models, and one with hybrid data, that includes additional information on asset ownership and education:

```
In [10]: #We now need to distinguish between 'traditional' and 'hybrid' variables for credit scoring
         #The hybrid contains all variables to calculate the informational value-add of alternative data
         traditional = ['defaulted',
          'code_gender',
          'amt_income_total',
          'days_birth',
          'days_employed',
          'name_income_type_Commercial associate',
          'name_income_type_Pensioner',
          'name_income_type_State servant',
          'name_income_type_Working']
         hybrid = final.columns.to_list()

         #we create two datasets to be used for model training to measure the difference in performance
         final_hybrid = final
         final_traditional = final[traditional]
```

We now need to think about the correct model to use here. I opt to use xgboost because it automatically takes care of a series of issues that could decrease the performance of models. Key advantages include:

- Regularization functionality to prevent overfitting
- Parallel processing for faster calculations in comparison to, for example, GBM
- Automatic handling of missing values
  Effective cross validation
- Effective tree pruning
- Limited vulnerability to multicollinearity issues as opposed to models such as logistic regression

In this way I hope to give each data foundation the best shot at having a model with the maximum predictive power.

We know that xgboost requires training, validation, and test data. Hence having created the datasets, we now split data into train (67%), validation (11%), and test sets (22%) for both traditional and hybrid data. I have chosen the proportions so that enough data is available for each step of the model development.

```
In [12]: #split data into train (67%), validation (11%), and test sets (22%) for both traditional and hybrid data

         from sklearn.model_selection import train_test_split

         #export hybrid data ensuring that label column is first
         data_dir = 'hybrid_credit_data'
         X = final_hybrid.drop('defaulted', axis=1)
         y = final_hybrid['defaulted']
         X_train_hybrid, X_test_hybrid, y_train_hybrid, y_test_hybrid = train_test_split(X, y, test_size=0.33, random_state=42)
         make_csv(y_train_hybrid, X_train_hybrid, filename='train.csv', data_dir=data_dir)
         make_csv(y_test_hybrid[:round(0.3*len(y_test_hybrid))], X_test_hybrid[:round(0.3*len(X_test_hybrid))], filename='valida
         pd.DataFrame(X_test_hybrid[round(0.3*len(X_test_hybrid)):]).to_csv(os.path.join(data_dir, 'test.csv'), header=False, in

         #export traditional data ensuring that label column is first
         data_dir = 'traditional_credit_data'
         X = final_traditional.drop('defaulted', axis=1)
         y = final_traditional['defaulted']
         X_train_traditional, X_test_traditional, y_train_traditional, y_test_traditional = train_test_split(X, y, test_size=0.3
         make_csv(y_train_traditional, X_train_traditional, filename='train.csv', data_dir=data_dir)
         make_csv(y_test_traditional[:round(0.3*len(y_test_traditional))], X_test_traditional[:round(0.3*len(X_test_traditional)
         pd.DataFrame(X_test_traditional[round(0.3*len(X_test_traditional)):]).to_csv(os.path.join(data_dir, 'test.csv'), header

         #NB: we have already made sure that the labels column (defauled) is first, as required by SageMaker
```

II.III. Benchmark

We have essentially 2 benchmarks. The first one is given by the previously mentioned study, from which we deduce that we need to maximize predictive performance subject to our false positive rate being below 4%. This essentially is a way to assess the applicability of our models to the real world. While helpful, we need to keep in mind that the data we have is not exactly the data that a real bank would have. Thus our distinction between traditional and hybrid data is more of a rough simulation than an exact comparison with the models used by real banks.

That is why our second benchmark is the comparison between the two models we are actually developing. Since we are interested in the informational value-add of alternative data we can compare the precision and accuracy of the traditional and hybrid model to see how many more people would be correctly labeled as worthy of a credit product if alternative data is considered, too.

III. Methodology

III.I. Data Preprocessing

We now use the AWS infrastructure and specifically the xgboost algorithm to build a model based on traditional and hybrid data. We set up a SageMaker session, fetch our standard execution role and create a bucket to store data in S3:

```
In [13]: #imports for SageMaker
         import boto3
         import sagemaker
```

```
In [14]: # session and role
         sagemaker_session = sagemaker.Session()
         role = sagemaker.get_execution_role()

         # create an S3 bucket
         bucket = sagemaker_session.default_bucket()
```

We then upload the relevant data in the respective S3 directories and ensure that the upload has been successful:

```
In [15]: #Uplad traditional data to S3
         data_dir = 'traditional_credit_data'
         prefix = 'traditional_credit_scoring'
         input_data = sagemaker_session.upload_data(path=data_dir, bucket=bucket, key_prefix=prefix)
         print(input_data)

         #Upload hybrid data to S3
         data_dir = 'hybrid_credit_data'
         prefix = 'hybrid_credit_scoring'
         input_data = sagemaker_session.upload_data(path=data_dir, bucket=bucket, key_prefix=prefix)
         print(input_data)

         s3://sagemaker-eu-central-1-303249258021/traditional_credit_scoring
         s3://sagemaker-eu-central-1-303249258021/hybrid_credit_scoring
```

In order to facilitate model comparisons later, I also wrote a simple model assessment function that returns the accuracy score for each model:

```
In [17]: #Create predictions with traditional model and assess model performance
         def model_assessment(data_dir, ground_truth):
             '''Function assesses model performance against ground truth data
                :param data_dir: data dictionary to retrieve predictions
                :param ground_truth: labels of test data set
                '''

             predictions = pd.read_csv(os.path.join(data_dir, 'test.csv.out'), header=None)
             predictions = [round(num) for num in predictions.squeeze().values]

             from sklearn.metrics import accuracy_score

             return accuracy_score(ground_truth, predictions)
```

III.II. Implementation

We now proceed to fit the models and make predictions. To ensure model comparability, we use the exact same model hyperparameters to fit the model:

```
#get xgb algorithm
container = get_image_uri(sagemaker_session.boto_region_name, 'xgboost')

#construct xgb estimator
xgb = sagemaker.estimator.Estimator(container,
                                    role,
                                    train_instance_count=1,
                                    train_instance_type='ml.m4.xlarge',
                                    output_path='s3://{}/{}/output'.format(sagemaker_session.default_bucket(), prefix),
                                    sagemaker_session=sagemaker_session)

#set xgb hyperparameters
xgb.set_hyperparameters(max_depth=5,
                        eta=0.2,
                        gamma=4,
                        min_child_weight=6,
                        subsample=0.8,
                        silent=0,
                        objective='binary:logistic',
                        early_stopping_rounds=10,
                        num_round=500)

train_location = sagemaker_session.upload_data(os.path.join(data_dir, 'train.csv'), key_prefix=prefix)
validation_location = sagemaker_session.upload_data(os.path.join(data_dir, 'validation.csv'), key_prefix=prefix)
test_location = sagemaker_session.upload_data(os.path.join(data_dir, 'test.csv'), key_prefix=prefix)

s3_input_train = sagemaker.s3_input(s3_data=train_location, content_type='csv')
s3_input_validation = sagemaker.s3_input(s3_data=validation_location, content_type='csv')

xgb.fit({'train': s3_input_train, 'validation': s3_input_validation})
```

We then use the transformer functionality to create predictions that can in turn be compared to the ground truth labels:

```
In [19]: #deploy traditional model
         xgb_transformer = xgb.transformer(instance_count = 1, instance_type = 'ml.m4.xlarge')
         xgb_transformer.transform(test_location, content_type='text/csv', split_type='Line')
         xgb_transformer.wait()
```

```
In [20]: #Save data from S3 locally
         !aws s3 cp --recursive $xgb_transformer.output_path $data_dir

         #Assess model
         print(model_assessment(data_dir=data_dir, ground_truth=y_test_traditional[round(0.3*len(X_test_traditional)):]))

         #Store predictions
         predictions = pd.read_csv(os.path.join(data_dir, 'test.csv.out'), header=None)
         predictions_traditional = [round(num) for num in predictions.squeeze().values]

         download: s3://sagemaker-eu-central-1-303249258021/xgboost-2020-06-22-15-15-07-437/test.csv.out to traditional_credit
         _data/test.csv.out
         0.8732782369146006
```

III.III. Refinement

Once we have completed the process for both models, we realize that an issue has come up that we considered before: the traditional model is 'cheating'. Checking its predictions we realize that it takes advantage of the unbalanced dataset and maximizes accuracy by simply predicting that every applicant will not default, thereby reaching an accuracy of close to 90% (i.e. the share of applicants who don't default). With this is does better than the hybrid model, but in practice it would be useless for a bank, since it does not help to distinguish between good and bad applicants. So we can note here a first if tentative win for the alternative data model - in some cases alternative data will be the only way to make any statistical inference about applicants if the availability of traditional data is sufficiently constrained:

```
In [23]: print('Number of people predicted to default based on traditional data is '+ str(predictions_traditional.count(1)))
         print('Number of people predicted to default based on hybrid data is '+ str(predictions_hybrid.count(1)))

Number of people predicted to default based on traditional data is 0
Number of people predicted to default based on hybrid data is 122
```

Having reviewed AWS' xgb documentation to find another evaluation metric that would have the algorithm focus more on the defaulted applicants, I couldn't find anything that would make the xgb based on traditional data deviate from its 'cheating' strategy. Hence, I now balance the dataset 50/50 to compare performance between the traditional and hybrid model. Since the data is now much smaller than our previous dataset, I opt for the Random Forest classifier, which is slimmer than the original xgboost and can be trained and deployed quicker while retaining many of the advantages of xgboost, including limited vulnerability to multicollinearity issues (which would be a problem in our dataset since we didn't drop one class in each of our multi-categorical variables):

```
Number of people predicted to default based on hybrid data is 122

In [37]: #create balanced dataset
         final_balanced = final.loc[final['defaulted']==0].sample(195).append(final.loc[final['defaulted'] != 0])

         #devide into hybrid and traditional data
         final_hybrid = final_balanced
         final_traditional = final_balanced[traditional]

In [38]: #split data into train (70%) and test sets (30%) for both traditional and hybrid data

         #export hybrid data ensuring that label column is first
         data_dir = 'hybrid_credit_data_balanced'
         X = final_hybrid.drop('defaulted', axis=1)
         y = final_hybrid['defaulted']
         X_train_hybrid, X_test_hybrid, y_train_hybrid, y_test_hybrid = train_test_split(X, y, test_size=0.3, random_state=42)

         #export traditional data ensuring that label column is first
         data_dir = 'traditional_credit_data_balanced'
         X = final_traditional.drop('defaulted', axis=1)
         y = final_traditional['defaulted']
         X_train_traditional, X_test_traditional, y_train_traditional, y_test_traditional = train_test_split(X, y, test_size=0.3

         #NB: we have already made sure that the labels column (defauled) is first, as required by SageMaker
```

IV. Results

IV.I. Model Evaluation and Validation

We are now ready to evaluate the model performance based on hybrid and traditional data. We will move in a 2-step sequence: we first use the confusion matrix to understand whether there is a significant difference along the major evaluation metrices such as recall, f1 score, and precision. If there is a substantial difference, we can draw an ROC curve to see compare the maximum accuracy of each model subject to the false positive rate being below 4%. We can then use the difference in these rates to calculate the number of additional applicants that would have gotten access to a financial product if hybrid data had been present. We can print out the classification report and confusion matrix for each of them:

In [39]:
```
#Test hybrid model
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report,confusion_matrix

dtree = RandomForestClassifier(random_state=0)
dtree.fit(X_train_hybrid, y_train_hybrid)
predictions = dtree.predict(X_test_hybrid)

#Import model evaluation metrics and print evaluation results
print(classification_report(y_test_hybrid,predictions))
print(confusion_matrix(y_test_hybrid,predictions))
```

```
              precision    recall  f1-score   support

           0       0.67      0.66      0.67        62
           1       0.60      0.61      0.60        51

   micro avg       0.64      0.64      0.64       113
   macro avg       0.63      0.63      0.63       113
weighted avg       0.64      0.64      0.64       113

[[41 21]
 [20 31]]
```

In [40]:
```
#test traditional model
dtree = RandomForestClassifier(random_state=0)
dtree.fit(X_train_traditional, y_train_traditional)
predictions = dtree.predict(X_test_traditional)

#Import model evaluation metrics and print evaluation results
print(classification_report(y_test_traditional,predictions))
print(confusion_matrix(y_test_traditional,predictions))
```

```
              precision    recall  f1-score   support

           0       0.64      0.66      0.65        62
           1       0.57      0.55      0.56        51

   micro avg       0.61      0.61      0.61       113
   macro avg       0.61      0.61      0.61       113
weighted avg       0.61      0.61      0.61       113

[[41 21]
 [23 28]]
```

In our case, the 0s are non-defaulted and 1s are defaulted applicants. We can see that the performance is very similar. The traditional model does slightly less well in correctly recognizing risky applicants and wrongly classifies 3 additional people as non-defaulted when in fact they would default. Given that we have overall 51 non-defaulted applicants this is an additional 5% of that population incorrectly classified. However, the models do equally well when classifying non-defaulted applicants.

IV.II. Justification

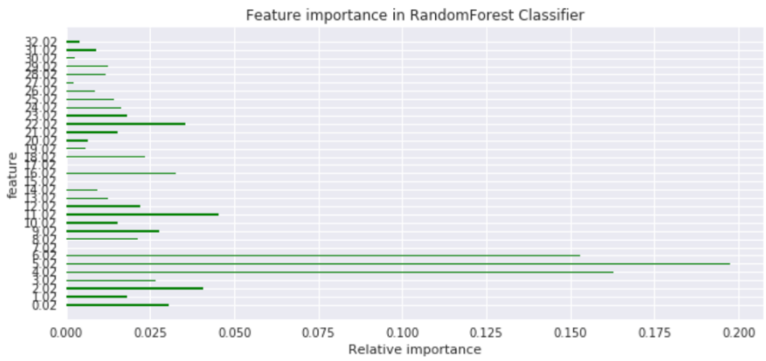
Correctly identifying applicants that won't default is the metric most relevant to us, since we try to understand how many more applicants would rightly get access to financial products with additional data. Unfortunately, we have little wiggle room to re-work the model. The rebalancing meant that we are already down to a few hundred observations and a further segmentation into distinct groups does not really make sense as a result. Since there is no substantial difference in performance, we do not carry out the ROC analysis and conclude that hybrid data does in this case lead to a higher fraction of deserving applicants being served.

V. Conclusions

V.I. Free-Form Visualization

If the two models perform equally well we should be able to see that the most important features are the ones that are included in both the traditional and hybrid model. In order to check this, we can visualize the importance of the various features:

```
col = [X_train_hybrid.columns.all()]
#modelname.feature_importance_
y = dtree.feature_importances_
#plot
fig, ax = plt.subplots()
width = 0.2 # the width of the bars
ind = np.arange(len(y)) # the x locations for the groups
ax.barh(ind, y, width, color='green')
ax.set_yticks(ind+width/10)
ax.set_yticklabels(col, minor=True)
plt.title('Feature importance in RandomForest Classifier')
plt.xlabel('Relative importance')
plt.ylabel('feature')
plt.figure(figsize=(10,5))
fig.set_size_inches(10.5, 4.5)
```



We can see that features 3-6 seems to be particularly important for the model. However, the features are listed in descending order, so we get a dictionary to understand how to translate the index into the feature:

	Features
0	code_gender
1	flag_own_car
2	flag_own_realty
3	cnt_children
4	amt_income_total
5	days_birth
6	days_employed
7	flag_mobil
8	flag_work_phone
9	flag_phone
10	flag_email
11	cnt_fam_members
12	name_income_type_Commercial associate
13	name_income_type_Pensioner
14	name_income_type_State servant
15	name_income_type_Student
16	name_income_type_Working
17	name_education_type_Academic degree
18	name_education_type_Higher education
19	name_education_type_Incomplete higher
20	name_education_type_Lower secondary
21	name_education_type_Secondary / secondary special
22	name_family_status_Civil marriage
23	name_family_status_Married
24	name_family_status_Separated
25	name_family_status_Single / not married
26	name_family_status_Widow
27	name_housing_type_Co-op apartment
28	name_housing_type_House / apartment
29	name_housing_type_Municipal apartment
30	name_housing_type_Office apartment
31	name_housing_type_Rented apartment

We can see that those are also exactly the features that we find in the traditional model, namely age, income, and job tenure.

V.II. Reflection

We set out to test whether additional data sources can help us increase the number of applicants that could benefit from financial products such as loans and credit cards if additional data was available on them. Based on the analysis and dataset we have here, we cannot draw this conclusion unequivocally. While we were successful at demonstrating that in some cases hybrid data is the only means of building any predictive model to support staff in making loans decision we did not prove that with a balanced dataset the informational value-add of data on education, asset ownership, and income is in any form significant.

V.III. Improvement

We should note that this is an effort to simulate and approximate a process that in real life works somewhat differently. Our dataset is taken from Kaggle, which gives us some degree of trust in its authenticity, but the quality of the data could not be independently verified. Moreover, even if correct, the available data set does not include some key information that banks might have in a real case scenario such as balance information on accounts, credit card information, deposits, and so on. We can see that the models and perhaps even data are non-representative of the real world because of the rather poor predictive performance of both models. In the first xgboost run, our hybrid model does actually worse than a completely naïve model that would simply predict that everyone is a non-defaulted applicant. In the second random forest run, both models do slightly better than chance or a naïve model but their overall performance remains too low to merit any substantial adoption by bank. As a result, it would be fantastic to run a similar analysis in partnership with a financial institution to see whether with a better data foundation it would be possible to train a model that would perform better than whatever model is currently in place at the bank to assess creditworthiness.