Nanodegree Engenheiro de Machine Learning

Projeto final

Vinicius Ferreira Santos
26 de novembro de 2018

I. Definição

Visão geral do projeto

O Brasil atualmente passa por uma grave crise financeira que culminou em mais de 13 milhões de desempregados. Muitos desses desempregados estavam pensando em adquirir seu primeiro imóvel, mas, sem um contracheque para comprovar a renda, acabaram perdendo o poder de financiamento e se juntando a outros milhares que não conseguem adquirir um financiamento de imóvel, devido à falta de crédito no mercado financeiro. Pensando no problema de falta de crédito para milhares de pessoas ao redor do mundo e no poder de previsão que pode ser alcançado com aprendizado de máquina, a Home Credit liberou seus dados e pediu ajuda para montar um modelo capaz de ajudar as pessoas com históricos de crédito insuficientes ou inexistentes a conseguirem empréstimos.

Descrição do problema

Muitas pessoas lutam para obter empréstimos devido aos históricos de crédito insuficientes ou inexistentes e, infelizmente, essa população é frequentemente aproveitada por credores não confiáveis. *Pensando nesse problema,* a Home Credit se esforça para ampliar a inclusão financeira para a população sem banco, proporcionando uma experiência de empréstimo positiva e segura. Para garantir que essa população carente tenha uma experiência de empréstimo positiva, a Home Credit utiliza uma variedade de dados alternativos - incluindo informações de telecomunicações e transacionais - para prever as capacidades de reembolso de seus clientes.

Atualmente a Home Credit usa vários métodos estatísticos e de aprendizado de máquina para realizar suas previsões. Fazer isso garantirá que os clientes com capacidade de pagamento não sejam rejeitados e que os empréstimos sejam

concedidos com um calendário principal, de vencimento e de reembolso, que capacitará seus clientes a serem bem-sucedidos.

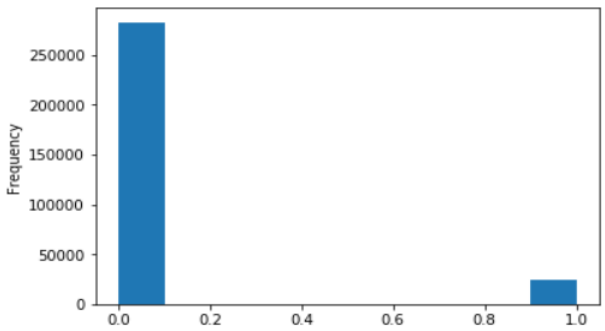
O principal objetivo desse projeto é prever quais pessoas sem crédito bancário terão capacidade de pagar seus empréstimos. Como esse é um tipo de problema que envolve o reconhecimento de padrões em uma grande massa de dados, o mesmo só pode ser resolvido, com certa assertividade, utilizando aprendizado de máquina, especificamente, aprendizado de máquina supervisionado de classificação.

Aprendizado de máquina tem sido amplamente utilizado no mercado, devido a facilidade e barateamento dos meios de armazenamento de dados. É uma subárea da inteligência artificial que se desenvolveu a partir da ideia de que os sistemas podem reconhecer padrões e tomar decisões, sem serem explicitamente programados. Problemas parecidos com o estudado aqui já estão sendo desenvolvidos e aperfeiçoados, como a detecção de fraudes [1] [2] [3], que são amplamente utilizadas no mercado financeiro.

Será através do aprendizado de máquina, que utilizarei os dados para treinar um modelo que informará se o cliente está apto ou não a receber um empréstimo. Após o modelo treinado, o mesmo poderá ser utilizado para prever clientes futuros, com a possibilidade de acompanhar o desempenho das previsões através de métricas.

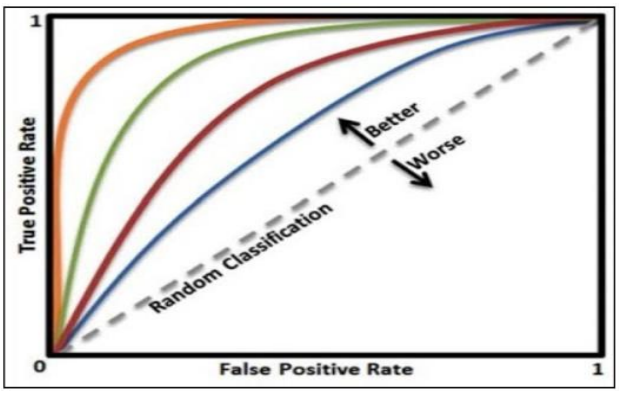
Métricas

A escolha da métrica que será utilizada neste projeto é baseado em um problema muito comum quando estamos trabalhando com classificação, **classes desbalanceadas**. Dentro dos dados de treinamento que serão utilizados, existem uma coluna chamada **TARGET**, que indica se um empréstimo foi pago a tempo (valor = 0) ou se o se o cliente teve dificuldade para realizar os pagamentos (valor = 1).



Como podemos perceber na imagem ao lado, nossos dados possuem mais informações de bons pagadores do que mal pagadores. Para esse problema em particular, é recomendado utilizar a métrica ROC AUC, que basicamente mede a taxa de exemplos positivos, quando eles

realmente são positivos **TP** (True Positive) e a taxa de exemplos positivos, quando na verdade eles são negativos **FP** (False Positive).

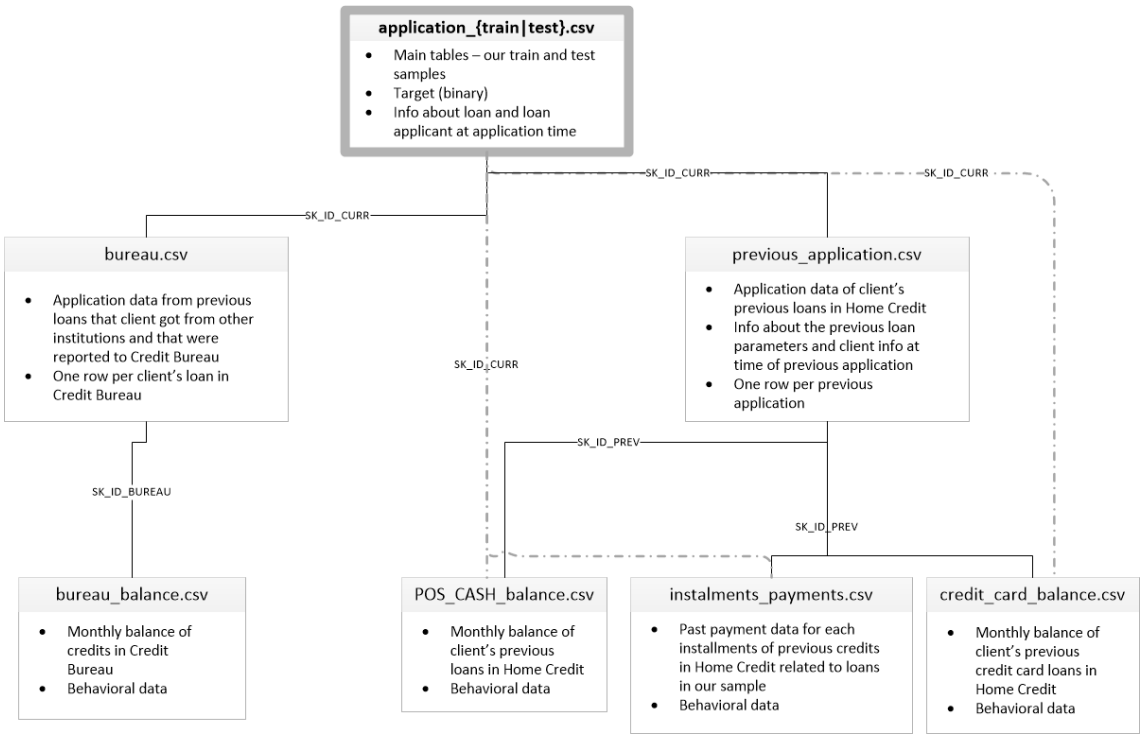


A imagem ao lado mostra como a métrica funciona, basicamente os resultados que estão acima da linha pontilhada estão obtendo resultados melhores, caso contrário, estão prevendo pior que a seleção aleatória.

II. Análise

Exploração dos dados

Como pode ser visto no diagrama de dados abaixo, os dados possuem diversas origens como dados de outras instituições(bureau.csv), mas para treinar o modelo que será utilizado no projeto, utilizarei somente os dados do arquivo application.csv de treino e teste. Os dados disponibilizados pela Home Credit estão disponíveis no kaggle.



Descrição de dados

- application_{train|test}.csv
    - This is the main table, broken into two files for Train (with TARGET) and Test (without TARGET).
    - Static data for all applications. One row represents one loan in our data sample.
- bureau.csv
    - All client's previous credits provided by other financial institutions that were reported to Credit Bureau (for clients who have a loan in our sample).
    - For every loan in our sample, there are as many rows as number of credits the client had in Credit Bureau before the application date.
- bureau_balance.csv
    - Monthly balances of previous credits in Credit Bureau.
    - This table has one row for each month of history of every previous credit reported to Credit Bureau – i.e the table has (loans in sample of relative previous credits of months where we have some history observable for the previous credits) rows.
- POS_CASH_balance.csv
    - Monthly balance snapshots of previous POS (point of sales) and cash loans that the applicant had with Home Credit.
    - This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to

loans in our sample – i.e. the table has (#loans in sample * # of relative previous credits * # of months in which we have some history observable for the previous credits) rows.

- credit_card_balance.csv
    - Monthly balance snapshots of previous POS (point of sales) and cash loans that the applicant had with Home Credit.
    - This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample * # of relative previous credits * # of months in which we have some history observable for the previous credits) rows.
- credit_card_balance.csv
    - Monthly balance snapshots of previous credit cards that the applicant has with Home Credit.
    - This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample * # of relative previous credit cards * # of months where we have some history observable for the previous credit card) rows.
- previous_application.csv
    - All previous applications for Home Credit loans of clients who have loans in our sample.
    - There is one row for each previous application related to loans in our data sample.
- installments_payments.csv
    - Repayment history for the previously disbursed credits in Home Credit related to the loans in our sample.
    - There is a) one row for every payment that was made plus b) one row each for missed payment.
    - One row is equivalent to one payment of one installment OR one installment corresponding to one payment of one previous Home Credit credit related to loans in our sample.
- HomeCredit_columns_description.csv
    - This file contains descriptions for the columns in the various data files.

O arquivo application.csv possui 30511 linhas, onde cada linha representa 1 mutuário, que tem suas informações distribuídas nas 120 colunas existentes como podemos ver na imagem abaixo:

	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_T
0	Cash loans	M	N	Y	0	202500.0

1 rows × 120 columns

Os dados possuem 3 tipos, dos quais 2 são números (float64 e int64) e o terceiro tipo é object, que representa dados categóricos. Os dados numéricos do tipo **float64** são compostos por 65 colunas, das quais somente 4 estão com todos as linhas preenchidas e existem 38 colunas com mais da metade das linhas sem informações. Já os dados numéricos do tipo **int64**, possuem 41 colunas e todas estão totalmente preenchidas. Os dados categóricos possuem 16 colunas, das quais 6 possuem dados ausentes.

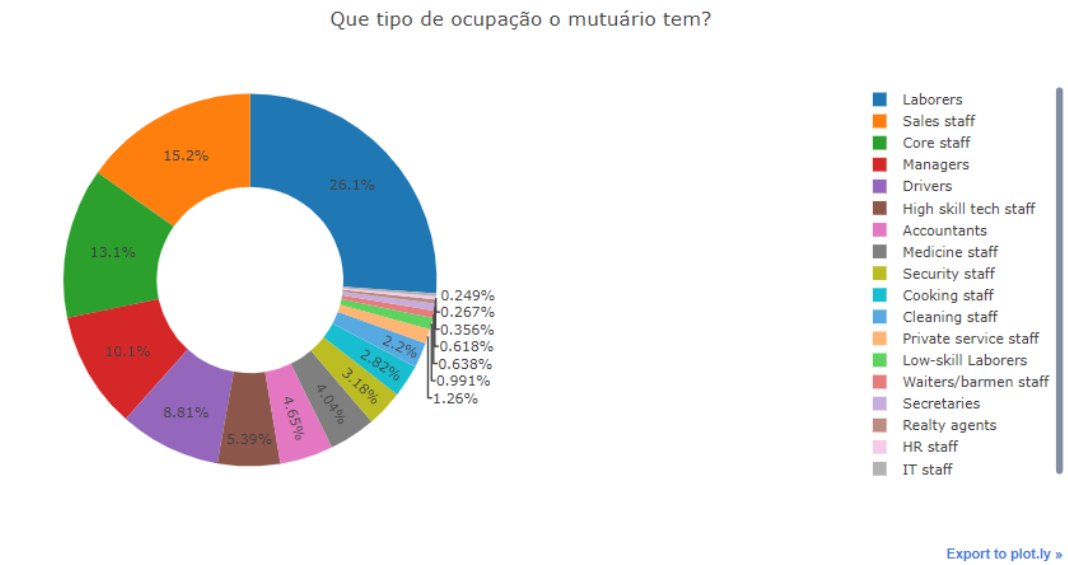
Outro ponto a ser notado nos dados é a presença de outliers, em alguns casos até aberrantes, como o maior valor encontrado na coluna **DAYS_EMPLOYED,** onde o maior valor possível deveria ser 0, mas o maior valor encontrado foi 365243, um valor positivo muito elevado que se dividirmos por 365 corresponderá a mais de mil anos.

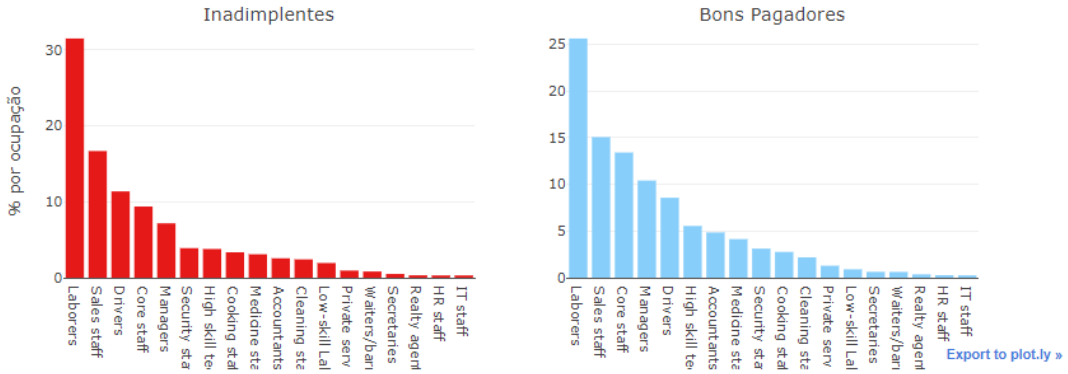
	CNT_CHILDREN	DAYS_BIRTH	DAYS_EMPLOYED	DAYS_ID_PUBLISH
count	307511.000000	307511.000000	307511.000000	307511.000000
mean	0.417052	-16036.995067	63815.045904	-2994.202373
std	0.722121	4363.988632	141275.766519	1509.450419
min	0.000000	-25229.000000	-17912.000000	-7197.000000
25%	0.000000	-19682.000000	-2760.000000	-4299.000000
50%	0.000000	-15750.000000	-1213.000000	-3254.000000
75%	1.000000	-12413.000000	-289.000000	-1720.000000
max	19.000000	-7489.000000	365243.000000	0.000000

Visualização exploratória

Como estamos tratando da possibilidade de fornecer empréstimos para pessoas sem crédito bancário, achei interessante mostrar primeiro algo relacionado a forma de trabalho dessas pessoas, que na grande maioria das vezes trabalham de maneira informal, mas não deixam de quitar suas dívidas. O cargo ocupado pela pessoa pode ser motivo para discriminação velada e, até mesmo, impedir a possibilidade de conseguir um empréstimo.

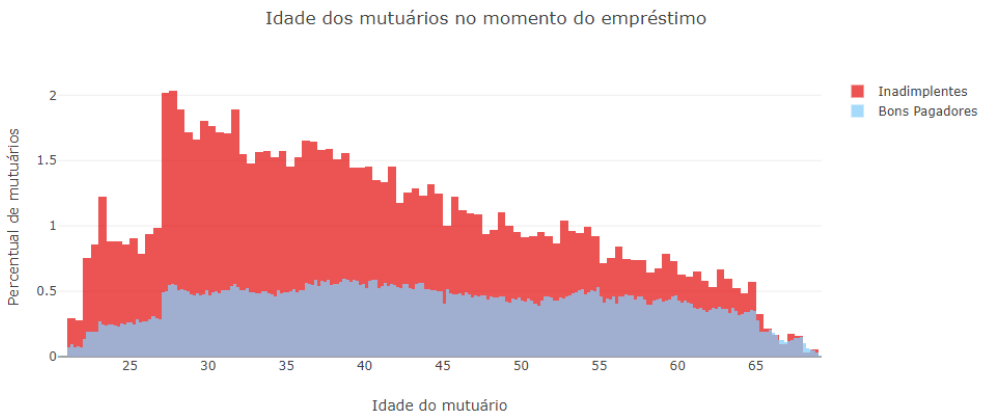
O gráfico abaixo mostra o tipo de ocupação de cada mutuário. Ao lado direito é possível visualizar a legenda que indica as cores das ocupações.



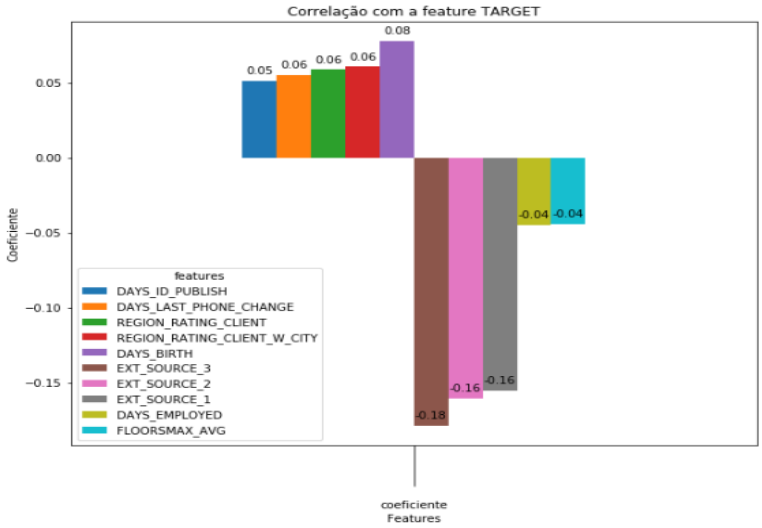


Aqui os 2 maiores percentuais são de trabalhadores sazonais: construção civil e vendedores. Certamente esses são dois perfis de mutuários que os credores tradicionais evitam, porque não conseguem manter uma estabilidade nos ganhos durante o ano. Podemos perceber que eles são os que mais pagam e os que mais devem, mas devemos lembrar que estamos trabalhando com classes desbalanceadas.

No gráfico abaixo é mostrado a distribuição da idade por situação do mutuário. Ao visualizar a sua distribuição, percebemos a relação direta do impacto da idade na inadimplência dos empréstimos. Existe uma queda gradual no número de mutuários inadimplentes à medida que a idade aumenta. Essa descoberta é de grande importância, porque sabemos que os bancos dificultam a liberação de empréstimos para os clientes idosos.

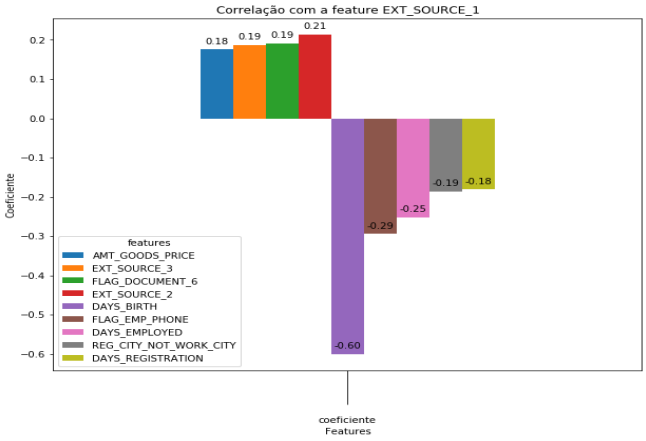


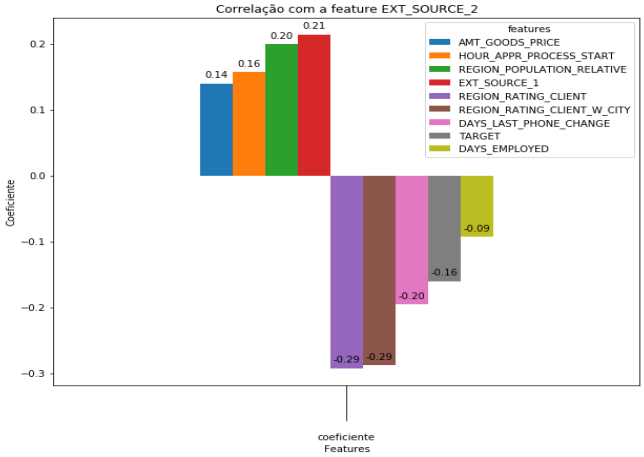
Como pretendo descobrir se um mutuário é inadimplente ou não, calculei o coeficiente de correlação de cada feature com o **TARGET**, que caso possua o valor 0, significaria que o mutuário conseguiu pagar o empréstimo a tempo e 1 significaria que teve problemas para cumprir com os pagamentos.

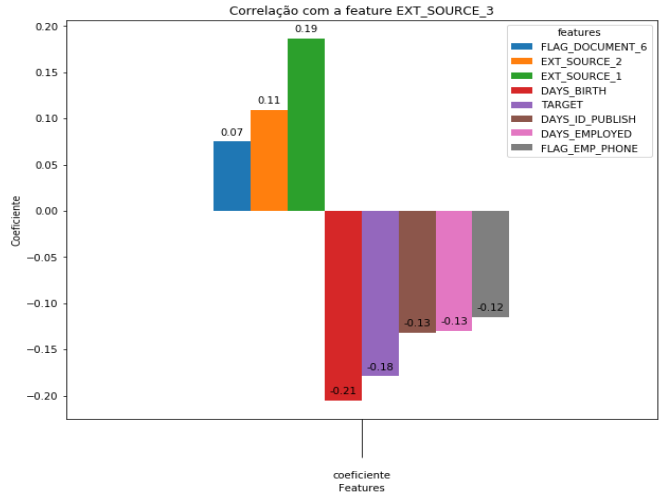


Como podemos perceber na visualização acima, as features EXT_SOURCE_1, EXT_SOURCE_2 e EXT_SOURCE_3 possuem as maiores correlações com a feature TARGET. O dicionário de dados indica que o conteúdo dessas features vêm de fontes de dados externos, ou seja, essa pontuação é calculada levando em consideração informações provindas de diversos conjunto de dados e que depois são normalizadas, isso explica os valores relativamente baixos.

Após descobrir quais eram as features mais correlacionadas com TAREGT, me aprofundei na análise das correlações e tentei entender mais sobre as 3 features mais correlacionadas: EXT_SOURCE_1, EXT_SOURCE_2 e EXT_SOURCE_3.







Visualizando as correlações acima, podemos identificar algumas coisas interessantes, mas como não sabemos o que compõe esses scores, só podemos fazer suposições. Podemos perceber que a feature EXT_SOURCE_1 possui maior correlação com a feature DAYS_BIRTH, isso mostra que de alguma forma essa feature faz parte da construção desse Score. Outra observação é que todas as outras correlações negativas com a feature EXT_SOURCE_1, são referentes as

informações relacionadas ao emprego do mutuário, com exceção da feature DAYS_REGISTRATION.

A feature EXT_SOURCE_2 aparenta ter mais interesse nos dados cadastrais de endereço do mutuário como: REGION_RATING_CLIENT, REGION_RATING_CLIENT_W_CITY e DAYS_LAST_PHONE_CHANGE; já EXT_SOURCE_3 aparenta ser uma junção da feature EXT_SOURCE_1 e EXT_SOURCE_2.

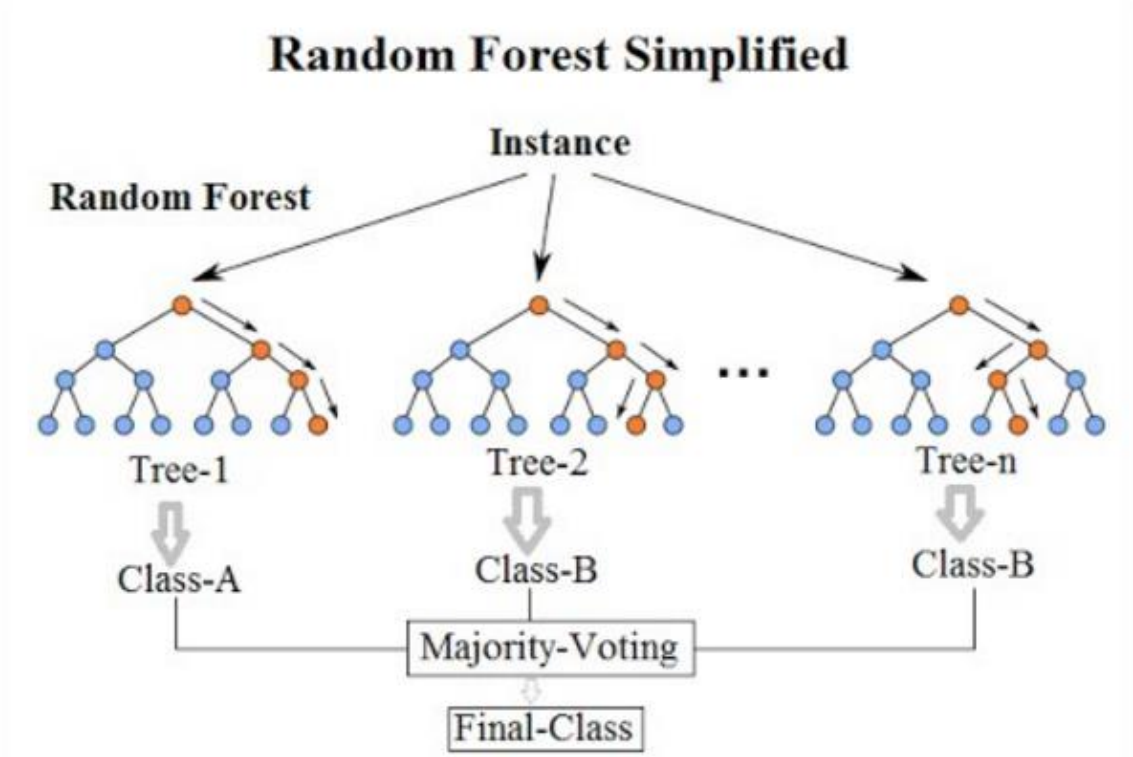
Algoritmos e técnicas

Como estou com um problema clássico de aprendizagem supervisionado de classificação e, além disso, com classes desbalanceadas, a possível solução deverá contemplar algoritmos e técnicas que minimizem e/ou solucionem tais problemas. Antes de adentrar nos méritos dos algoritmos e técnicas, para trabalhar em um cenário supervisionado, primeiramente precisaremos de uma grande quantidade de dados; depois, os dados precisam ter um indicativo do que estamos buscando, ou seja, os dados precisam indicar se determinado empréstimo foi pago ou não. Felizmente, a Home Credit forneceu os dados para treinamento do algoritmo e para os devidos testes. Os dados de teste só serão utilizados quando tiver um modelo treinado, a fim de realizar testes mais realistas para saber se o modelo desenvolvido está conseguindo generalizar bem com dados nunca visto.

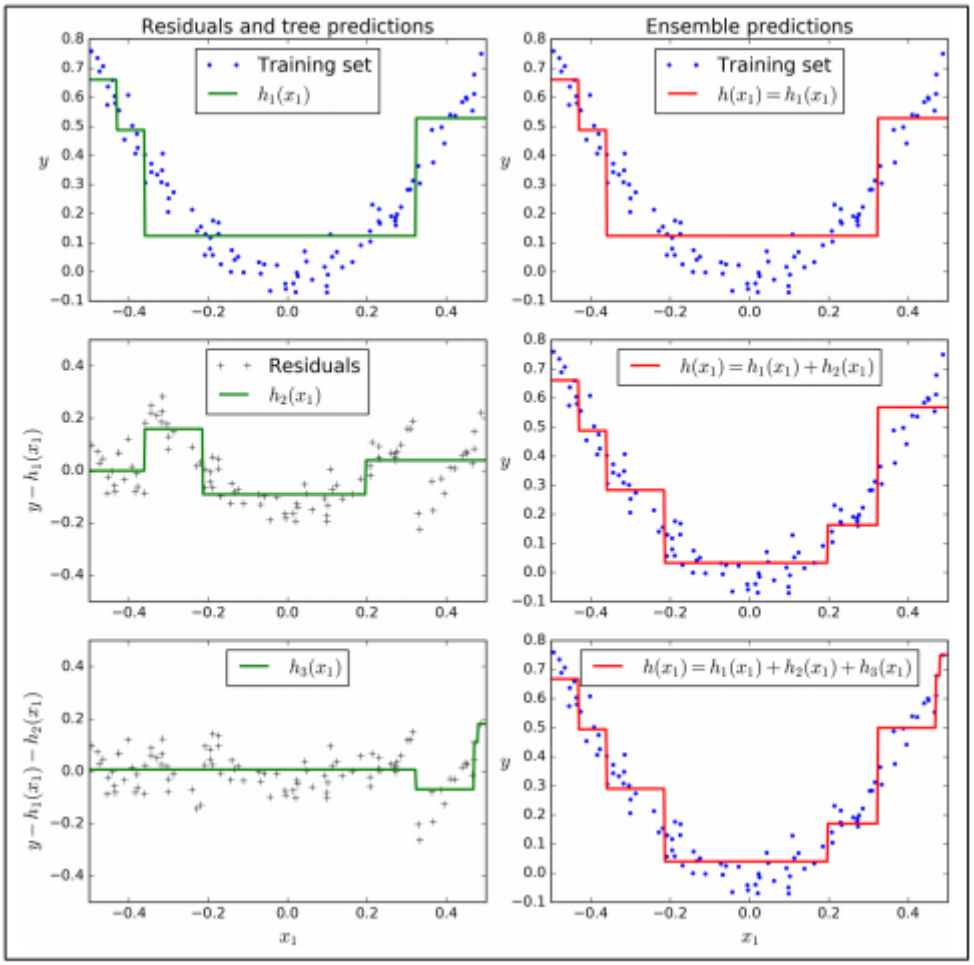
O conjunto de treinamento será quebrado em dois, onde o primeiro conterá somente a feature **TARGET**, que, como sabemos, indica se o empréstimo foi pago ou não. O segundo conjunto conterá somente os atributos que usaremos para realizar o treinamento. O conjunto de teste não possui a feature **TARGET**, por isso não sofrerá o mesmo tipo de separação.

Para preparar os dados para os algoritmos de aprendizado de máquina, ao invés de fazer manualmente, irei escrever algumas funções e/ou classes para tratar de problemas comuns nos conjuntos de dados. Dessa maneira, poderei reproduzir as técnicas em dados novos sem muito trabalho. Os algoritmos que serão utilizados, foram escolhidos porque são algoritmos de natureza supervisionada de classificação e porque são robustos o suficiente para tratar dos problemas das classes desbalanceadas. Outro ponto que indica o uso é a possibilidade de verificar a importância das features, coma essa informação em mãos será possível eliminar features que não possuem muito significado para os modelos, já que muitas features, aumenta a probabilidade de causar overfitting. Serão utilizados os seguintes algoritmos:

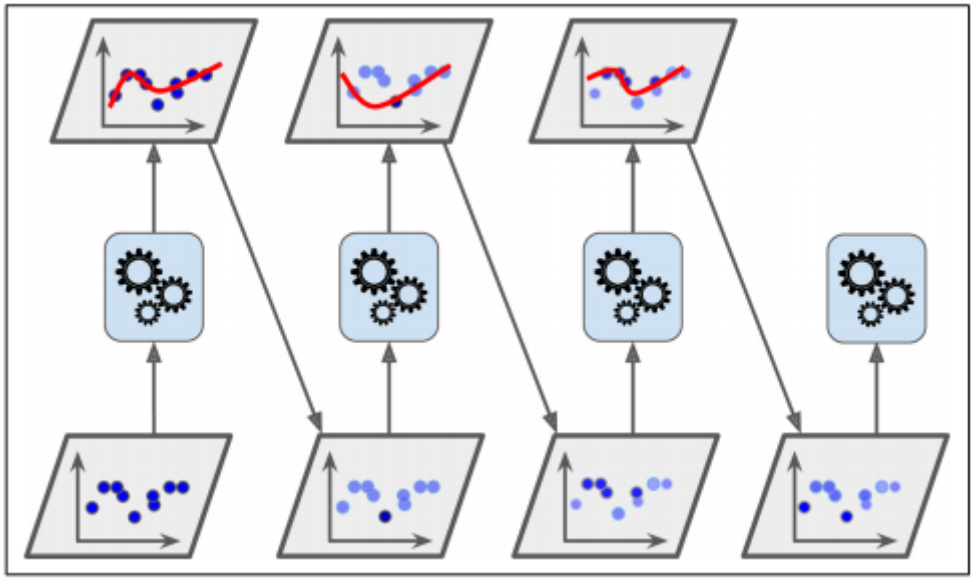
- RandomForestClassifier – é um algoritmo de aprendizado supervisionado, responsável por criar uma floresta de *arvores de decisão*, que são treinadas, na maioria das vezes, usando o método *bagging*. O método *bagging* parte do princípio que a previsão de um modelo pode ser impreciso, mas uma combinação de modelos aumenta o aprendizado geral.



  Na Random Forest, em cada arvore de decisão, cada nó interno representa uma resposta sobre uma feature, cada ramificação representa o resultado dessa resposta e cada nó folha representa um rótulo de classe (decisão tomada após computar todos as features). Em termos leigos, é a mesma coisa que se pedíssemos opiniões a diversas pessoas e ao coletar todas as respostas, conseguíssemos sintetizar e sugerir um caminho a pessoas que tivessem o interesse na mesma pergunta.
- GradientBoostingClassifier – é um algoritmo de aprendizado supervisionado utilizando o método *boosting*, que se refere a um grupo de algoritmos que utilizam médias ponderadas para tornar os modelos fracos em modelos mais fortes. O GradientBoosting constrói árvores de decisão uma de cada vez, onde cada nova árvore ajuda a corrigir erros cometidos por árvores previamente treinadas. Só que em vez de ajustar os pesos de instância em cada iteração, esse método tenta ajustar a nova árvore de decisão aos erros residuais feitos pela árvore anterior.



- AdaBoostClassifier – é um algoritmo de aprendizado supervisionado, que utiliza uma combinação das ideias de Bagging e Boosting. Adaboost pode ser utilizado para melhorar o desempenho de qualquer algoritmo de aprendizado de máquina. A ideia por trás do Adaboost, é que a cada iteração os pesos de cada exemplo classificado incorretamente será aumentado (ou alternativamente, os pesos classificados corretamente são decrementados), para que então o novo classificador trabalhe em mais exemplos.



Por exemplo: o peso relativo das instâncias de treinamento classificadas incorretamente é então aumentado. Um segundo classificador é treinado usando os pesos atualizados e, novamente, faz previsões sobre o conjunto de treinamento, os pesos são atualizados e assim por diante, como pode ser percebido na imagem acima.

Os 3 algoritmos serão testados com as configurações padrões e o que apresentar o melhor desempenho será selecionado. Após identificar o melhor algoritmo, o próximo passo será realizar uma otimização, que consistem em escolher o melhor conjunto de hiperparâmetros que elevem a probabilidade de prever em qual classe o empréstimo se encaixará. Para essa atividade, será realizada uma pesquisa exaustiva utilizando GridSearchCV, em um conjunto de parâmetros que será definido manualmente.

Benchmark

Como modelo de referência, usarei uma instância do RandomForestClassifier com os valores padrões. Para comparar os resultados, o modelo final selecionado deverá ultrapassar o valor base do RandomForestClassifier utilizando a métrica selecionada.

III. Metodologia

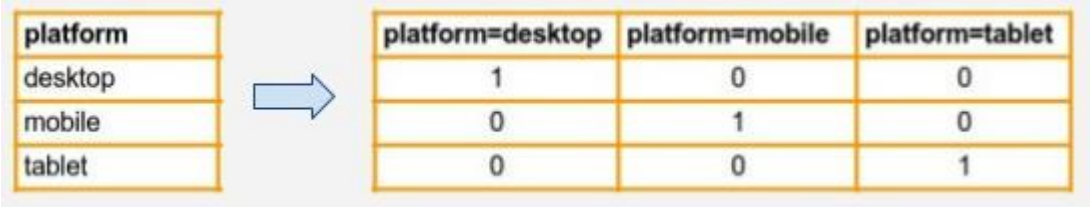
Pré-processamento de dados

A maioria dos algoritmos de aprendizado de máquina precisam de alguns requisitos para transmitir confiança nos seus resultados, entre eles estão: dados de alta qualidade e conhecimento sobre o negócio, afim de extrair informações importantes escondidas nas features existentes. Para tentar atingir esses objetivos é preciso, quase sempre, de um pré-processamento nos dados, dessa maneira, o processo de pré-processamento dos dados de treinamento e teste que realizei seguirá os seguintes passos:

1. Precisamos nos preocupar se existem problemas relacionados com a qualidade dos dados, como dados ausentes e outliers:
    a. **Outliers**: usarei o Método Turco para identificar valores atípicos, onde *o valor discrepante* é maior/menor em 1,5 vezes a variação interquartil (IQR). Qualquer valor que se encaixe nesse cenário será substituído por *NaN*.
    b. **Valores Ausentes**: Para tratar dos valores ausentes, usarei a seguinte regra: irei remover as features que apresentarem mais que 50% de dados ausentes, afim de minimizar possíveis erros na classificação.
2. É necessário realizar algumas transformações/criações em algumas features de domínio, por exemplo, as features de valores financeiros são log-normal por padrão, dessa maneira, criarei novas features adicionando "_LOG" ao final da descrição para indicar que é uma feature transformada. Outras features que sofrerão uma transformação são as que representam datas, por padrão, elas têm seus valores contados em dias. Para facilitar as análises, serão criados novas features com os valores em anos, mas manterei as antigas em dias.
3. Quase sempre uma combinação de features produz outras mais correlacionados que as originais com a feature TARGET, sendo assim, serão realizadas algumas combinações como: valor do crédito por renda, renda do mutuário por idade, crédito por idade e etc.
4. Serão criadas features polinomiais de 3º grau nas mais correlacionados com TARGET; como estamos tratando de emprestar dinheiro, também serão usadas as features financeiras AMT_CREDIT e AMT_INCOME_TOTAL. Essas features que serão geradas são uma combinação de todas as selecionadas. Às vezes, essas features combinadas se tornam correlacionadas com TARGET¸ coisa que não acontecia com elas separadas. A ideia central aqui é tentar descobrir se as combinações das features financeiras produzem uma correlação forte com o TARGET**.**
5. Como já tínhamos vários valores ausentes e acrescentei muito mais ao remover os outliers, será imputado a mediana nas features numéricas, devido as diferentes distribuições das features.
6. A grande maioria dos algoritmos não executam bem com dados numéricos com diferentes escalas. Dessa maneira, será utilizado

StandardScaler, que basicamente vai subtrair o valor de uma feature pela sua média e dividir o resultado pelo desvio padrão do conjunto de dados.

7. Outro problema é que a maioria dos algoritmos de aprendizado de máquina não funcionam bem com dados textuais, como temos várias features categóricas e elas são textos, será utilizado o *OneHotEncoder*, que basicamente transformará as features categóricas em numéricas, como mostrado abaixo:



platform
desktop
mobile
tablet

platform=desktop	platform=mobile	platform=tablet
1	0	0
0	1	0
0	0	1

Implementação

Foram utilizados três algoritmos de aprendizado de máquina: RandomForestClassifier, GradientBoostingClassifier e AdaBoostClassifier. Como foi dito anteriormente, estes foram escolhidos porque são algoritmos de natureza supervisionada de classificação e porque são robustos o suficiente para tratar dos problemas das classes desbalanceadas. Em todos os casos, os dados precisaram serem pré-processados antes de serem utilizados, pensando nisso, senti a necessidade de uma estrutura mais robusta para reproduzir as transformações tantos nos dados de treino quanto nos de teste. Dessa maneira, acabei criando algumas funções e classes para facilitar o processo.

Para tratar dos outliers, usei a seguinte função:

```
def remove_outliers(df_numeric, df):
    for feature in df_numeric.keys():
        # Calcula Q1 (25º percentil dos dados)
        Q1 = np.percentile(df_numeric[feature], 25)
        # Calcula Q3 (75º percentil dos dados)
        Q3 = np.percentile(df_numeric[feature], 75)
        # Utiliza a amplitude interquartil para calcular o valor discrepante (1,5 vezes a variação interquartil)
        step = (Q3 - Q1) * 1.5
        index = df[~((df[feature] >= Q1 - step) & (df[feature] <= Q3 + step))].index
        #atribui nan nas linhas com valores discrepantes
        df.loc[df.index[index], feature] = np.nan
```

A função acima irá identificar os valores considerados outliers e irá trocar o seu valor por **nan**. Após tratar dos outliers, precisei remover os valores ausentes, para isso, removi os valores que apresentaram mais de 50% de ausência, mas essa remoção acabou deixando os dados de treino e testes com número de features diferentes. Para solucionar esse problema, precisei alinhar os conjuntos de dados, mantendo somente as features que existiam nos 2 conjuntos, como mostrado abaixo:

```
features_train = features_train.loc[:, features_train.isnull().mean() < .5]
#Alinhando as colunas de teste com as de treinamento
features_train, features_test = features_train.align(features_test, join = 'inner', axis = 1)
```

O próximo passo, foi criar uma função que tratasse da criação/conversão das features financeiras para log-normal e das datas em dias para anos:

```
def transformando_e_criando_features(df):
    # recuperando nome das features categóricas
    cat_attribs = nomes_atributos_categoricos(df)
    # converte as colunas object em str
    df[cat_attribs] = df[cat_attribs].astype(str)
    # variáveis financeiras tem uma distribuição log-normal
    df['AMT_INCOME_TOTAL_LOG'] = np.log(df['AMT_INCOME_TOTAL'])
    df['AMT_CREDIT_LOG'] = np.log(df['AMT_CREDIT'])
    df['AMT_ANNUITY_LOG'] = np.log(df['AMT_ANNUITY'])
    #criando colunas e transformando datas em dias para anos
    df['AGE'] = (abs(df['DAYS_BIRTH']) / 365).astype(np.int64)
    df['YEAR_EMPLOYED'] = (abs(df['DAYS_EMPLOYED']) / 365).fillna(0).astype(np.int64)
    df['YEAR_REGISTRATION'] = (abs(df['DAYS_REGISTRATION']) / 365).fillna(0).astype(np.int64)
```

Para a criação das features combinadas, polinomiais e categóricas, precisei criar as classes abaixo:

```
class PipelineBuilder():

    def build_cat_pipeline(self, cat_attribs):
        return Pipeline([
                ('selector', DataFrameSelector(cat_attribs)),
                ('cat_encoder', OneHotEncoder(sparse=False)),
            ])

    def build_numeric_pipeline(self, num_attribs,
                               poly_features=None,
                               combined_features = None):
        return  Pipeline([
                    ('selector', DataFrameSelector(num_attribs)),
                    ('imputer', Imputer(strategy = "median")),
                    ('attribs_adder', CombinedAttributesAdder(poly_features=poly_features,
                                                              combined_features = combined_features)),
                    ('std_scaler', StandardScaler()),
                ])

    def build_full_pipeline(self, cat_attribs, num_attribs,
                            poly_features=None,
                            combined_features = None):
        return FeatureUnion(transformer_list=[
                    ("num_pipeline", self.build_numeric_pipeline(num_attribs,
                                                                 poly_features=poly_features,
                                                                 combined_features = combined_features)),
                    ("cat_pipeline", self.build_cat_pipeline(cat_attribs)),
                ])

    def build_full_pipeline_with_predictor(self, clf, cat_attribs,
                                           num_attribs, poly_features=None,
                                           combined_features = None):
        return Pipeline([
                ("preparation", self.build_full_pipeline(cat_attribs, num_attribs,
                                                         poly_features=poly_features,
                                                         combined_features = combined_features)),
                ("clf", clf)
            ])
```

PipelineBuilder

```python
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

DataframeSelector

```python
class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, poly_features=None, combined_features = None):
        self.poly_features = poly_features
        self.combined_features = combined_features
        self.name_combined_features = []
        self.name_created_poly_features = []
        self.created_poly_features = None
    def fit(self, X, Y=None):
        return self
    def transform(self, X, Y=None):
        if np.any(self.poly_features):
            poly_transformer = PolynomialFeatures(degree = 3)
            for _, value in self.poly_features.items():
                # recupera as features que serão transformadas
                poly_features = X[:, value[1]]
                poly_transformer.fit(poly_features)
                self.created_poly_features = poly_transformer.transform(poly_features)
                #recupera o nome das features criadas
                self.name_created_poly_features = poly_transformer.get_feature_names(input_features = value[0])
            X = np.c_[X, self.created_poly_features]
        if np.any(self.combined_features):
            for key, value in self.combined_features.items():
                #recupera o nome da combinação desejada
                self.name_combined_features.append(key)
                #divide a feature da 1º posição da lista pela feature da 2º posição
                new_feature = X[:, value[0]] / X[:, value[1]]
                X = np.c_[X, new_feature]
        return X
```

CombinedAttributesAdder

Para facilitar as transformações, criei a classe *PipelineBuilder*, que, como o nome já diz, é responsável por criar as pipelines utilizadas nas criações/transformações das features nas ordens informadas. A classe *DataframeSelector* é responsável por selecionar as features através dos nomes informados; já a classe *CombinedAttributesAdder*, é responsável por realizar as combinações e transformações polinomiais.

```python
pipelineBuilder = PipelineBuilder()
full_pipeline = pipelineBuilder.build_full_pipeline(cat_attribs, num_attribs,
                                                    poly_features=poly_features,
                                                    combined_features = combined_features)
#transformando as features de treinamento
features_attr_combinados_train = full_pipeline.fit_transform(features_train)
#recuperando os nomes das features polinomias criadas
poly_names = full_pipeline.transformer_list[0][1].named_steps['attribs_adder'].name_created_poly_features
#recuperando os nomes das features combinadas criadas
combined_names = full_pipeline.transformer_list[0][1].named_steps['attribs_adder'].name_combined_features
#recuperando os nomes das features categóricas após onehotencoding
cat_encoder = full_pipeline.transformer_list[1][1].named_steps["cat_encoder"]
cat_names = []
for itens, feature in zip(cat_encoder.categories_, cat_attribs):
    for item in itens:
        cat_names.append('{}_{}'.format(feature, item))
#refazendo o dataframe com os nomes das features
data_attr_combinados_train = pd.DataFrame(
    features_attr_combinados_train,
    columns=list(features_train[num_attribs].columns)+poly_names
                + combined_names
                + cat_names)
```

Exemplo de utilização da solução nos dados de treino

Após realizar a transformação nos dados de treino, a pipeline devolve todas as features criadas/transformadas, mas precisei realizar as mesmas transformações nos dados de teste, só que antes disso, é preciso utilizar o *Imputer* treinado para preencher os dados ausentes no conjunto de teste, se não estaremos imputando valores diferentes, já que são dois conjuntos com dados diferentes. Ao transformar os dados de teste, apareceu novamente o problema de features que existem em treino e não em teste, isso já era esperado, principalmente porque utilizei *OneHotEncoder* nas features categóricas e pode acontecer de algumas categorias existirem em treino e não em teste ou vice-versa, o que ocasiona o número diferente de features após a transformação. Para esse resolver esse problema, foi preciso alinhar novamente os dois conjuntos de dados, mantendo as features em comum.

Com os dados de treino já pré-processados, foi utilizado *train_test_split*, estratificando os dados em cima dos valores da feature TARGET, garantindo a proporção para cada classe e evitando um viés desnecessário, já que nossas classes são desbalanceadas.

```
x_train, x_test, y_train, y_test = train_test_split(features, labels,
                                                    test_size=6000,
                                                    random_state=42,
                                                    stratify=labels)
```

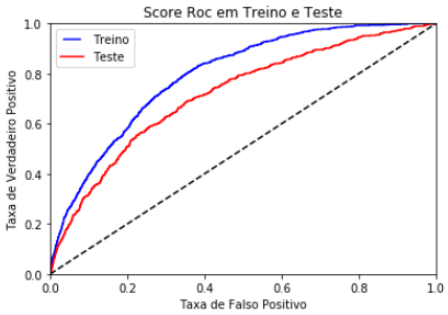
Para testar o desempenho dos algoritmos, foi utilizado a métrica ROC AUC que é mais indicada para trabalhar com classes desbalanceadas e para facilitar a visualização dos resultados, foi criado a função *draw_roc_score*, que sintetiza várias informações necessárias para o entendimento do desempenho dos modelos.

```python
def draw_roc_score(classifier, x_train, y_train, x_test, y_test):
    #train
    y_probas_train = classifier.predict_proba(x_train)
    y_scores_train = y_probas_train[:, 1]
    predictions_train = classifier.predict(x_train)
    cv_score_train = cross_validation.cross_val_score(classifier, x_train, y_train, cv=5, scoring='roc_auc')
    print("Train Accuracy: {}%".format(accuracy_score(y_train, predictions_train)))
    print("Train Score Roc: {}%".format(np.round(roc_auc_score(y_train, y_scores_train)*100, 3)))
    print("Train CV Score : Mean - %.7g | Std - %.7g | Min - %.7g | Max - %.7g" % (np.mean(cv_score_train),
                                                                                   np.std(cv_score_train),
                                                                                   np.min(cv_score_train),
                                                                                   np.max(cv_score_train)))
    print("----------------------------------------------------------------------------------------------")
    #teste
    y_probas_test = classifier.predict_proba(x_test)
    y_scores_test = y_probas_test[:, 1]
    predictions_test = classifier.predict(x_test)
    cv_score_test = cross_validation.cross_val_score(classifier, x_test, y_test, cv=5, scoring='roc_auc')
    print("Test Accuracy: {}%".format(accuracy_score(y_test, predictions_test)))
    print("Test Score Roc: {}%".format(np.round(roc_auc_score(y_test, y_scores_test)*100, 3)))
    print("Test CV Score : Mean - %.7g | Std - %.7g | Min - %.7g | Max - %.7g" % (np.mean(cv_score_test),
                                                                                  np.std(cv_score_test),
                                                                                  np.min(cv_score_test),
                                                                                  np.max(cv_score_test)))
    #draw roc curve
    fpr1, tpr1, thresholds = roc_curve(y_train, y_scores_train)
    fpr2, tpr2, thresholds = roc_curve(y_test, y_scores_test)
    plt.title("Score Roc em Treino e Teste")
    line1, = plt.plot(fpr1, tpr1, 'b', label="Treino", linewidth=1.5)
    line2, = plt.plot(fpr2, tpr2,'r', label="Teste", linewidth=1.5)
    plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('Taxa de Falso Positivo')
    plt.ylabel('Taxa de Verdadeiro Positivo')
    plt.show()
```

Resultados da função:

```python
classifier = AdaBoostClassifier()
t0 = time()
classifier.fit(x_train, y_train)
print("Tempo de execução Adaboost:", round(time()-t0, 3), "s")
draw_roc_score(classifier, x_train, y_train, x_test, y_test)
```

```
Tempo de execução Adaboost: 3.083 s
Train Accuracy: 0.9194%
Train Score Roc: 79.307%
Train CV Score : Mean - 0.717786 | Std - 0.01726962 | Min - 0.6923086 | Max - 0.7465727
----------------------------------------------------------------------------------------------
Test Accuracy: 0.9155%
Test Score Roc: 71.711%
Test CV Score : Mean - 0.6796754 | Std - 0.02830649 | Min - 0.6277808 | Max - 0.7087716
```



Refinamento

Os resultados dos algoritmos com os valores padrões estão presentes na tabela abaixo:

Algoritmo	ROC AUC em Treino %	Accuracy em Treino %	ROC AUC em Teste %	Accuracy em Teste %
RandomForestClassifier	99,99	98,54	60,29	91,80
GradientBoostingClassifier	85,58	92,45	73,12	91,61
AdaBoostClassifier	79,30	91,94	71,71	91,55

Tabela 1

Para selecionar o melhor modelo utilizando a métrica ROC AUC, optei pelo que conseguisse a maior pontuação nos dados de teste, dessa maneira, com base na tabela acima, podemos perceber que o modelo com o algoritmo GradientBoostingClassifier obteve a melhor pontuação com 73,12%.

Para realizar a otimização, utilizei a seguinte regra: escolher a menor diferença possível entre os resultados de treino e teste de ROC AUC, garantindo assim, que o modelo otimizado está conseguindo generalizar bem com dados nunca visto. No processo de otimização, procurei seguir com pequenos passos, selecionado poucos parâmetros em cada ajustes e definindo no passo seguinte, o melhor ajuste sugerido pela busca exaustiva do GridSearchCV do passo anterior.

O processo de otimização foi separado em 7 etapas, como podemos ver na tabela abaixo:

Etapas	Hiperparâmetro(s)	Melhor Ajuste	ROC AUC em Treino %	ROC AUC em Teste %	Diferença ROC treino - teste
1	n_estimators	90	90.83	71.97	18,86

2	max_depth e min_samples_split	3 e 900	81.54	73.08	8,46
3	min_samples_leaf	35	81.27	73.03	8,24
4	min_samples_leaf e min_samples_split	10 e 1700	80.58	73.42	7,16
5	max_features	24	80.57	74	6,57
6	learning_rate	0,1	80,10	73,89	6,21
7	Subsample	0,9	80.70	73.71	6,99

Tabela 2

Como podemos perceber na tabela acima, os ajustes realizados na etapa 6 apresentaram os melhores resultados e por isso, o melhor modelo sugerido na etapa 6 foi selecionado como o modelo final.

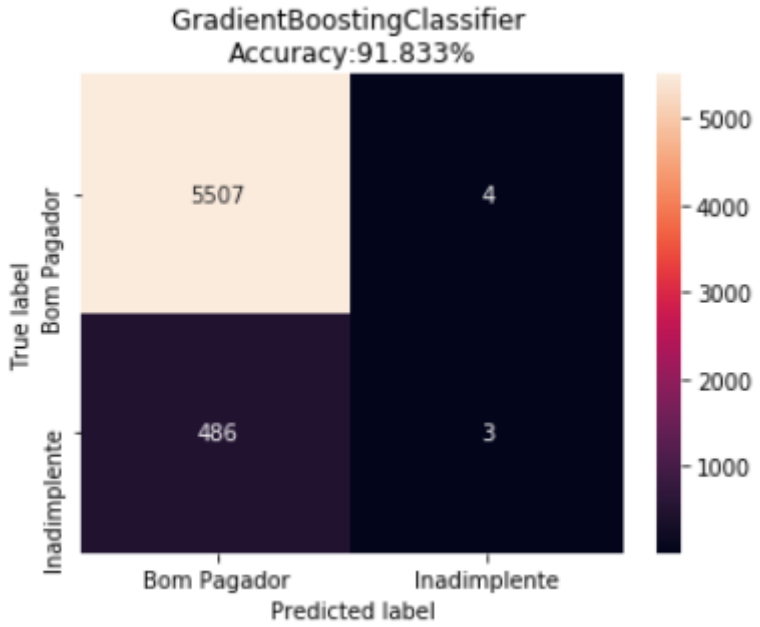
IV. Resultados

Modelo de avaliação e validação

O propósito do modelo de aprendizado de máquina desenvolvido aqui era prever se um futuro mutuário irá pagar seu empréstimo ou não. A partir do momento que optei por utilizar somente 1 fonte de dados das 7 fornecidas pela Home Credit, eu sabia que seria complicado construir um modelo que tivesse um desempenho razoável, porque acreditava que as features mais importantes estariam divididas nos 7 arquivos. Pensando nisso, no processo de otimização utilizei os seguintes critérios para chegar ao modelo final: o primeiro ponto é que existiam grandes diferenças entre os 3 modelos nos hiperparâmetros min_samples_leaf, min_samples_split e subsample, onde, para o min_samples_leaf e min_samples_split, o ideal é que se escolham os valores baixos para controlar o over-fitting, já para subsample, o ideal é que o valor seja menor que 1 com a finalidade de reduzir a variação. O segundo ponto é a diferença entre o score roc

de treino e teste, onde a menor diferença representa que o modelo está generalizando bem para dados não vistos. Dessa maneira, levando em consideração os pontos citados anteriormente, os modelos encontrados nas etapas **5 e 6** apresentaram os melhores ajustes. O modelo do passo 5 apresenta os melhores ajustes definidos no primeiro ponto e o 6 apresenta a menor diferença entre os scores, dessa forma, irei ficar o passo 6 por generalizar melhor.

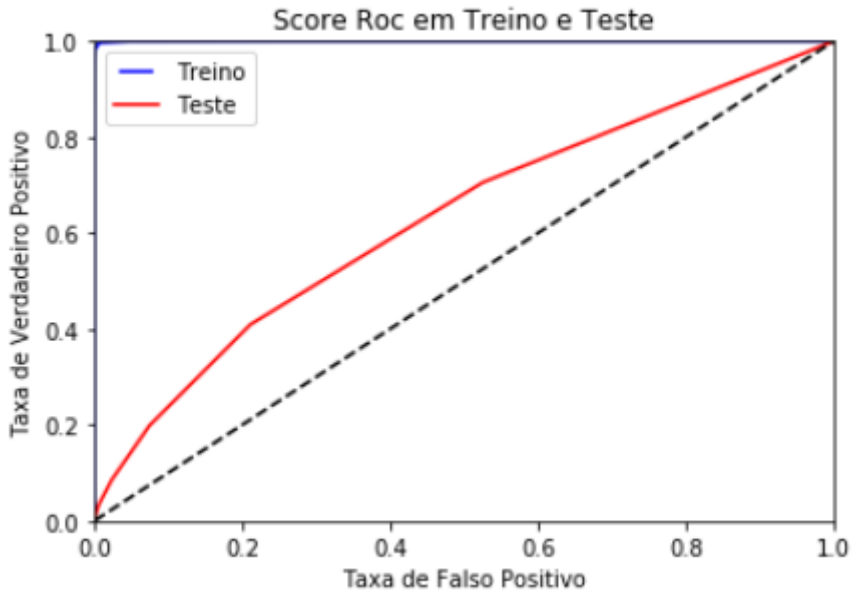
Após construir a solução final, realizei testes em dados que ainda não tinham sido utilizados no treinamento e me deparei com o resultado abaixo:



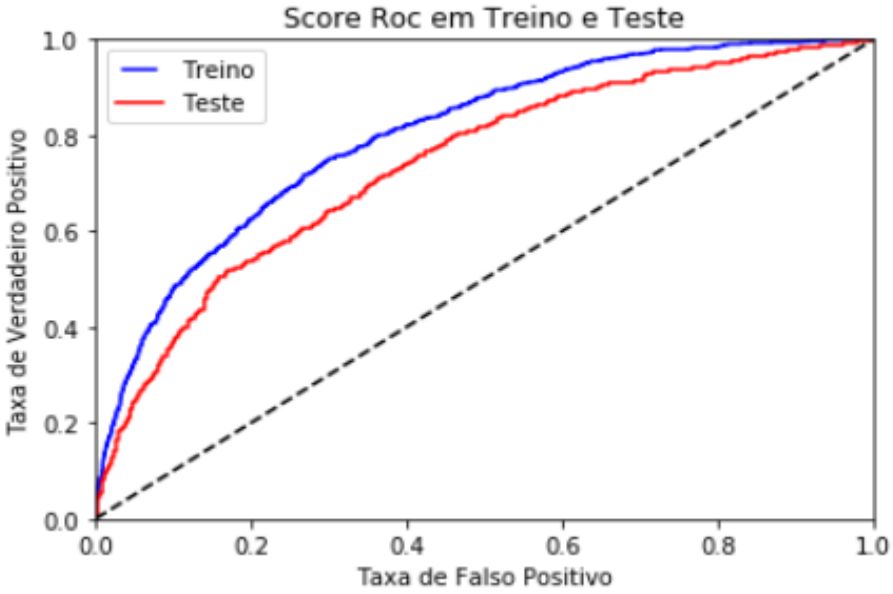
Como podemos perceber na imagem acima, o classificador possui uma precisão de 91,8% na identificação da situação dos mutuários, ou seja, ele está errando em 8,2% das vezes em que foi utilizado. O modelo classificou 5507 mutuários como bons pagadores que realmente era bons pagadores, 4 mutuários que eram bons pagadores como inadimplentes, 486 bons pagadores quando na verdade eram inadimplentes e 3 inadimplentes que realmente eram inadimplentes. Dessa maneira, acredito que o modelo desenvolvido, apesar dos poucos erros apresentados, é confiável ao ponto de ser utilizado como fonte de **primeira consulta** para empréstimo a pessoas sem crédito bancário.

Justificativa

Para analisar o desempenho do modelo final, irei comparar com o modelo de benchmark.



BenchMark



Modelo Final

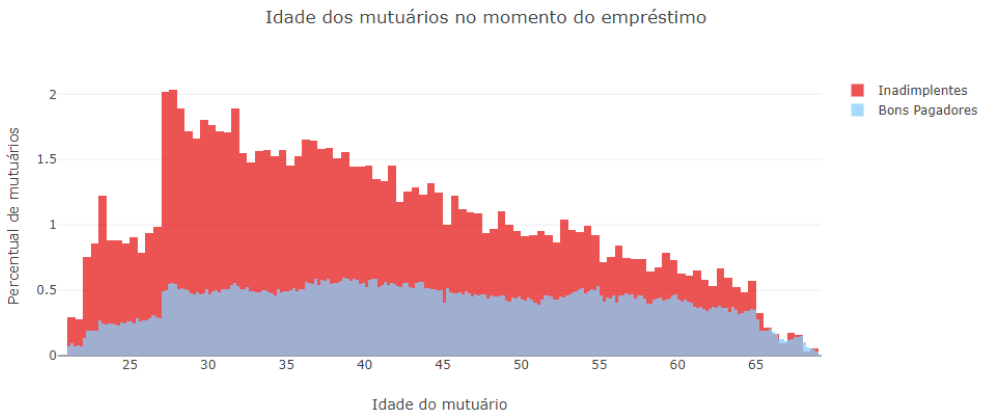
Olhando para a imagem do benchmark, podemos facilmente ver que o resultado apresenta **overfitting,** os resultados em treinamento chegaram a 99,98% enquanto em teste apresentaram 62,99%, com uma diferença de 28%. O algoritmo fez o que pôde com os dados que tinham no momento e conseguiu se sair um pouco melhor que uma seleção aleatória.

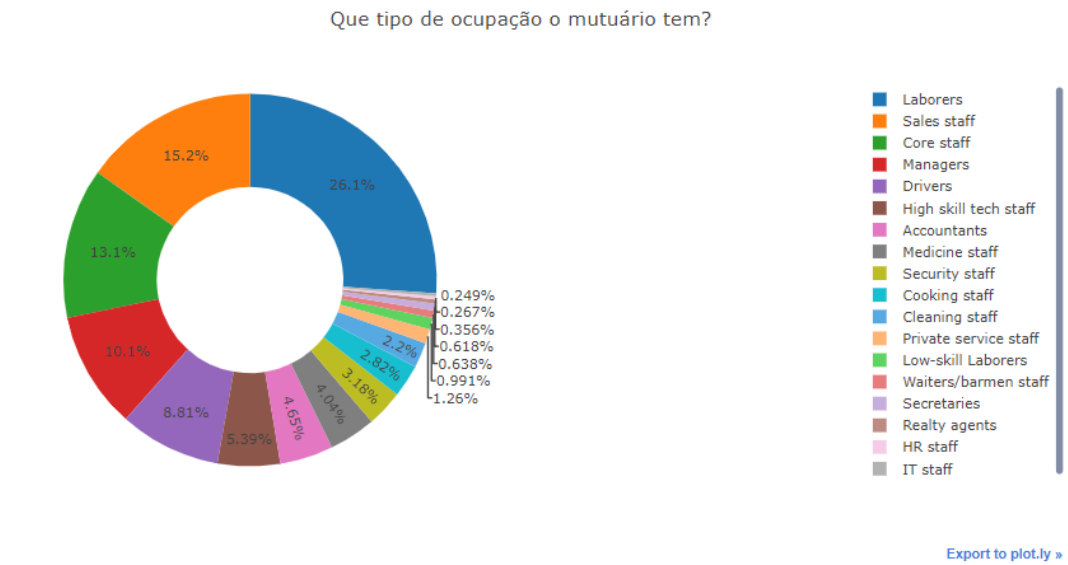
A imagem do modelo final apresenta uma melhora significativa nos resultados, tanto nos dados de treinamento como nos de teste. Como vimos na tabela 2, o modelo apresentou um resultado de 80,10% em treino e 73,89 em teste, com uma diferença de 6,21%. Os dados que o modelo final utilizou já possuía muito mais features e uma qualidade superior comparado ao que o de benchmark utilizou. Com resultados tão expressivos, não existe dúvida que os ajustes utilizados, estão sendo determinantes na melhor assertividade do modelo final.

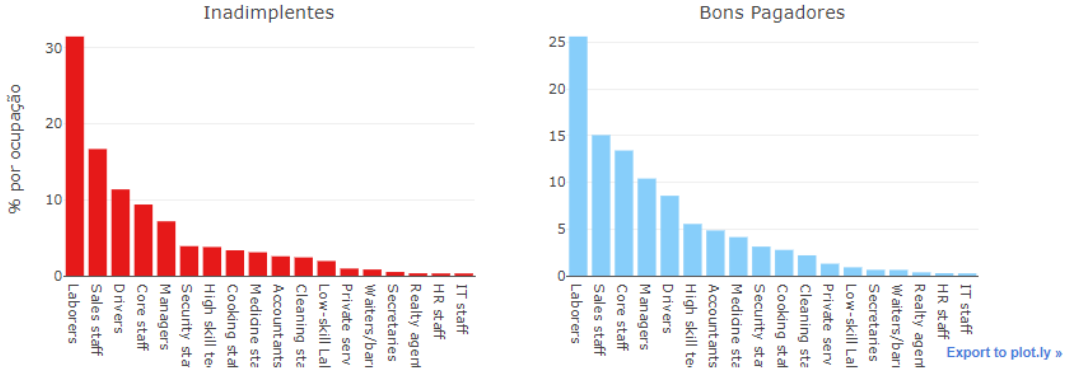
V. Conclusão

Forma livre de visualização

A análise exploratória foi crucial para o melhor desenvolvimento do projeto, foi através das daquelas descobertas que pude entender como funcionava cada feature e saber qual caminho seguir no momento do pré-processamento. Acredito que o ponto alto das visualizações, foi a descoberta que os idosos tinham mais responsabilidade com os pagamentos que os mais jovens como podemos visualizar na imagem abaixo. Normalmente, os idosos são rejeitados por bancos e instituições não bancárias que trabalham com empréstimos, mas como pudemos visualizar na seção de visualização exploratória, os mais jovens são mais inadimplentes que os idosos. Essa descoberta, junto com a descoberta que os trabalhadores da construção civil (pedreiros e etc) são responsáveis nos seus pagamentos, poderia causar um impacto enorme nos lucros dessas instituições, se eles fornecessem mais condições para esses tipos de clientes.







Reflexão

A tentativa de prever o melhor cliente para conceder um empréstimo, é o principal objetivo das empresas do ramo financeiro e este tipo de problema pode ser solucionado utilizando algoritmos de aprendizado de máquina (machine learning). Escolhi esse projeto, porque é um tema de estrema necessidade para a situação atual do Brasil e porque já fui questionado por pessoas ligadas a instituições financeiras na minha cidade, sobre soluções para problemas parecidos.

Para chegar a produtos passíveis de serem utilizados na prática, é preciso que os resultados apresentem uma segurança e demonstre seus pontos fortes e fracos, porque esse é um tipo de trabalho que dificilmente vai passar uma certeza absoluta. Para minimizar essas incertezas, em meio ao problema das classes desbalanceadas, foram utilizados algoritmos especialistas em classificação. Os dados precisaram de um pré-processamento para remover problemas de baixa

qualidade e para tentar encontrar novas features que aumentasse o potencial de previsão.

Todos os algoritmos utilizados são da biblioteca Scikit-learn (RandomForestClassifier, GradientBoostingClassifier e AdaBoostClassifier), passando mais segurança, devido a todos terem sidos extensivamente testados. O RandomForestClassifier foi utilizado como benchmark, mas como estava utilizando os dados praticamente sem ajustes nenhum, foi logo superado pelos outros dois. O GradientBoostingClassifier foi selecionado por apresentar o melhor desempenho e foi amplamente melhorado com o uso do GridSearchCV, na tentativa de encontrar o melhor conjunto de hiperparâmetros que aumentasse o poder de previsão.

A parte mais difícil do projeto foi a otimização, nessa parte precisei buscar novos conhecimentos em artigos, livros e com alguns mentores para tentar encaixar o melhor ajuste no modelo, afim de não causar **overfitting**, prejudicando as previsões de futuros clientes.

Acredito que a solução desenvolvida para este projeto, pode ser levada em consideração por todos os tipos de negócios que realizam a liberação de crédito para futuros clientes, somente se baseando em comprovação de renda por vínculo empregatício.

Melhorias

Os resultados obtidos por esse projeto certamente podem ser melhorados, começando por acesso a mais dados. Como eu optei por utilizar somente uma das sete possíveis fontes de dados no projeto, as melhorias poderiam começar com o acesso a todos os dados possíveis, que gerariam diversas novas features possivelmente mais correlacionadas com a target. Indo mais além, outra opção para aumentar as informações sobre os clientes e consequentemente o potencial das previsões, seria coletar dados públicos nas redes sociais e fazer um cruzamento de informações, com utilização de visão computacional e processamento de linguagem natural, a fim de tentar entender os padrões de vida e consumo dos clientes.

Dentro do escopo do projeto, acredito que outra possível melhoria seria a utilização de PCA com o modelo final, para verificar se teria ganhos com a redução de dimensionalidade. Por fim, faria mais algumas tentativas de melhoria com a utilização de outros algoritmos e até mesmo redes neurais.

Referências

https://hackernoon.com/how-to-develop-a-robust-algorithm-c38e08f32201

https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd

https://github.com/rafaelmartinsbuck/credit-card-fraud-detection/blob/master/Credit_card_fraud_detection.ipynb

https://pt.wikipedia.org/wiki/Aprendizado_de_m%C3%A1quina

https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d

https://en.wikipedia.org/wiki/Random_forest#Algorithm

https://lamfo-unb.github.io/2017/09/27/BaggingVsBoosting/

https://www.quora.com/What-is-bagging-in-machine-learning

http://www.ceavi.udesc.br/arquivos/id_submenu/387/brigiane_machado_da_silva___marcos_vanderlinde.pdf