Projeto Final

Lucas Meneghesso Faria

27/03/2019

Nanodegree Engenheiro de Machine Learning

Kaggle Competition - Home Credit Default Risk

Definição

Visão geral do Projeto

O Projeto desenvolvido no presente documento se encontra no domínio de **fornecimento de crédito**. Este é um tema que historicamente possui problemas na distribuição de crédito a parcelas da população que não possuem histórico de crédito. As técnicas de Ciência de Dados podem ajudar a otimizar a análise de **risco de inadimplência**.

Foram realizadas etapas exploratórias e aplicadas técnicas de Aprendizado de Máquina com o objetivo de gerar um modelo preditivo de risco de inadimplência para novos clientes da empresa **Home Credit**.

Os dados com as informações (e consequentemente o perfil) dos clientes foram fornecidos via competição da plataforma Kaggle: *https://www.kaggle.com/c/home-credit-default-risk*.

Descrição do Problema

O problema escolhido foi baseado na competição da Kaggle: "**Home Credit Default Risk**". Foram fornecidos dados históricos sobre os clientes e o objetivo é usar estes dados para criar um modelo preditivo de forma a prever se os clientes serão capazes de pagar o crédito fornecido. Em outras palavras, o objetivo é prever a inadimplência dos clientes.

No passado os bancos enfrentavam o problema da **subjetividade nas análises de risco de crédito**, conforme apontado por Altman e Saunders (1998). Existem estudos anteriores ao presente trabalho que mostram o ganho obtido com a aplicação de técnicas de Ciência de Dados, como por exemplo o trabalho de Aniceto (2016). Neste trabalho é possível ver a evolução ao longo do tempo com técnicas estatísticas para análises, como **Regressão Logística**, **Modelo Probit**, **Análise Discriminante**.

Com o crescimento da utilização de modelos estatísticos e de aprendizado de máquina veio a melhora nas qualidades das análises e tomadas de decisões.

O problema pode ser expresso da seguinte forma:

- Com os dados históricos de clientes passados, criar um modelo preditivo para identificação da probabilidade de um cliente ser inadimplente.
- Dado um novo cliente, qual é a probabilidade P de ele ser inadimplente?
- P é maior do que um limite T escolhido? Caso positivo, não fornecer crédito para esse cliente.

A estratégia utilizada para criar o modelo preditivo descrito acima foi a seguinte, utilizando primeiramente os dados do arquivo **application_train.csv**:

- **Modelo Baseline**
- **Análise - application_train.csv:**
    - Análise Exploratória das features
        - Distribuições, NaNs, features categóricas, etc.
    - Impacto das técnicas de pré-processamento
        - Substituição de NaNs, tranformações das features, etc.
    - Engenharia de Features
        - Extração de informações com base nas features existentes
        - Redução de dimensionalidade via seleção de features informativas ou combinação de features
- **Criação de Modelos Preditivos Generalizáveis**
    - Utilização de técnicas de validação cruzada
    - Calibração de hiper-parâmetros
    - Criação de métodos de agregação de modelos
- **Reiniciar o ciclo em busca de melhora na performance**
- **Reiniciar o ciclo utilizando os outros arquivos fornecidos pela Kaggle em busca de melhora na performance**

A Figura 1 representa o fluxo descrito acima:

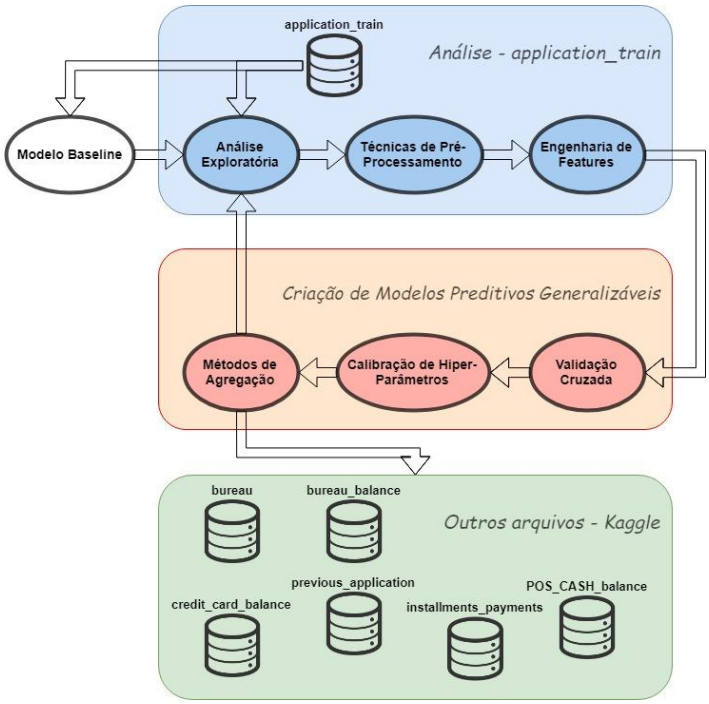

Figura 1. Diagrama – fluxo do projeto

Com relação ao modelo Baseline foram testados modelos lineares e baseados em árvore amplamente utilizados na comunidade de Ciência de Dados, de forma a escolher o que inicialmente teria a melhor performance sem nenhum pré-processamento (além do necessário para features categóricas). Os algoritmos testados serão discutidos mais a frente.

A Análise Exploratória foi realizada iniciando-se pelas features que o modelo Baseline considerou mais importantes. Isso permitiu a geração de idéias para a Engenharia de Features.

Com relação ao modelo final, foi utilizado um modelo que recebe as features idealizadas na etapa de Engenharia de Features e os seus hiperparâmetros foram tunados de forma a termos a melhor performance.

Métricas

A métrica utilizada para quantificar a qualidade do modelo foi a **AUCROC** (area abaixo da curva ROC), considerando a probabilidade prevista e se o **cliente foi inadimplente ou não**. Essa métrica é definida com base na curva ROC (Receiver Operating Characteristic) e é muito utilizada em problemas de fornecimento de crédito. Em um problema como esse estamos interessados em classificar o cliente como sendo um cliente com **alto risco de inadimplência** ou um cliente com **baixo risco**. Não estamos interessados em detectar o número exato da probabilidade do cliente ser inadimplente, mas sim na ordem da saída do modelo: clientes que realmente se tornaram inadimplentes **com score maior** do que clientes que não se tornaram inadimplentes. Em outras palavras isso configura um problema de **rankeamento**.

A AUCROC representa uma forma compacta de medir a qualidade do modelo preditivo em diferentes limites de probabilidade. Quanto maior o valor da AUCROC, maior é o poder do modelo em separar as **distribuições da classe alvo** (target = 1) das **distribuições da classe padrão** (target = 0) de acordo com as probabilidades previstas, conforme a Figura 2 mostra um modelo com AUC máxima igual a 1.

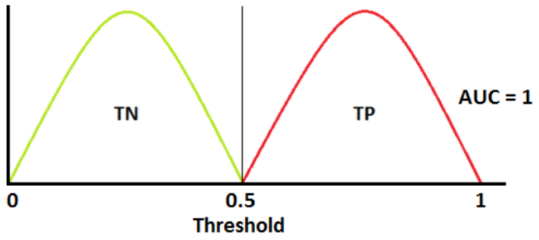

Figura 2. Distribuição Target = 0 vs Target = 1

Fonte: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

Foi também a métrica utilizada pelos organizadores da competição para avaliação dos modelos preditivos criados pelos competidores.

A função **cross_val_model** (arquivo evaluation.py) ajudou na comparação do poder preditivo entre os modelos criados durante o projeto. Como o objetivo destas avaliações não foi coletar uma estimativa da AUC, mas sim apenas comparar a performance dos modelos em diversas situações (diferentes modelos, diferentes conjuntos de features, etc), foram utilizados todos os dados de treinamento para essa finalidade. É importante ressaltar que esta função foi implementada utilizando a técnica de validação cruzada **K-fold**

estratificado, de forma a conservar a relação de clientes target=0/target=1 nas amostras. Foi utilizado o valor **random_state=10**, para podermos comparar os valores.

Análise

Exploração dos Dados

Foram considerados apenas os dados do arquivo **application_train.csv** como tratativa inicial pois este contém os dados principais dos clientes da própria Home Credit. A Figura 3 ilustra alguns exemplos dos dados. Como o número de features é muito grande, não foi possível mostrar todos os valores.

```
application_train.head()
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CRE
0	100002	1	Cash loans	M	N	Y	0	202500.00	40659
1	100003	0	Cash loans	F	N	N	0	270000.00	129350
2	100004	0	Revolving loans	M	Y	Y	0	67500.00	13500
3	100006	0	Cash loans	F	N	Y	0	135000.00	31268
4	100007	0	Cash loans	M	N	Y	0	121500.00	51300

5 rows × 122 columns

Figura 3. Exemplos de dados - application_train.csv

Foram analisados os seguintes quesitos:

- **Tamanho**:
  O conjunto de dados possui **307511** amostras e **122** colunas (121 features e um Target).

```
In [159]: print("application_train:     ",application_train.shape)
          application_train:      (307511, 122)
```

Figura 4. Tamanho do conjunto de dados.

- **Amostras duplicadas**:
  Foi identificado que o conjunto não possui amostras duplicadas utilizando a biblioteca Pandas.
- **Distribuição do Target**:
  Foi identificado que este é um conjunto de dados com alto grau de desbalanceamento, conforme a Figura 5 ilustra.

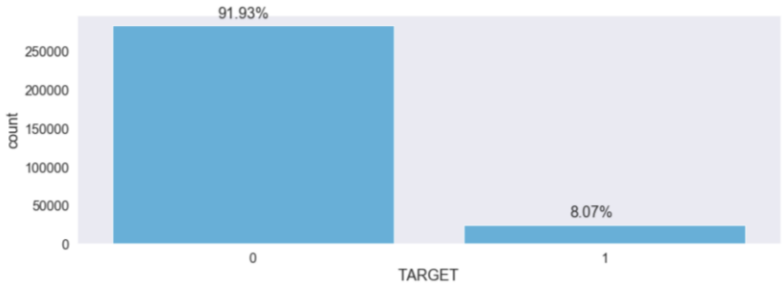

Figura 5. Distribuição do Target

- **Tipos de dados**:
  Foram identificadas features com valores categóricos e numéricos. Para facilidade no tratamento, foi utilizada a nomenclatura a seguir:
  - Feature binária: possui valores 0/1 ou N/Y.
  - Feature categórica: possui valores nominais/alfanuméricos.
  - Feature contínua: possui valores contínuos.
  - Feature ordinal: possui valores inteiros.

  A Figura 6 mostra a distribuição dos tipos de features.

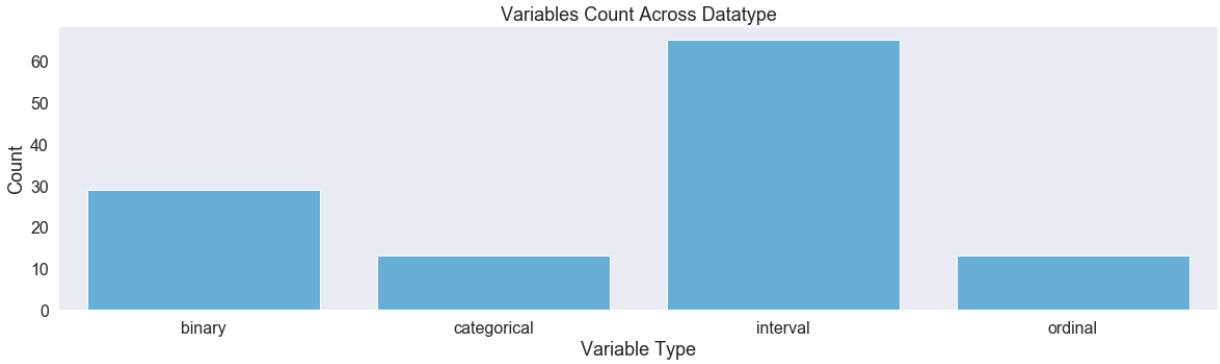

Figura 6. Tipos de features no arquivo application_train.csv

- **Valores não preenchidos**:
  Foi identificado que o conjunto de dados possui um alto número de valores não preenchidos. Isso descarta medidas simples, como a simples remoção de valores não preenchidos.

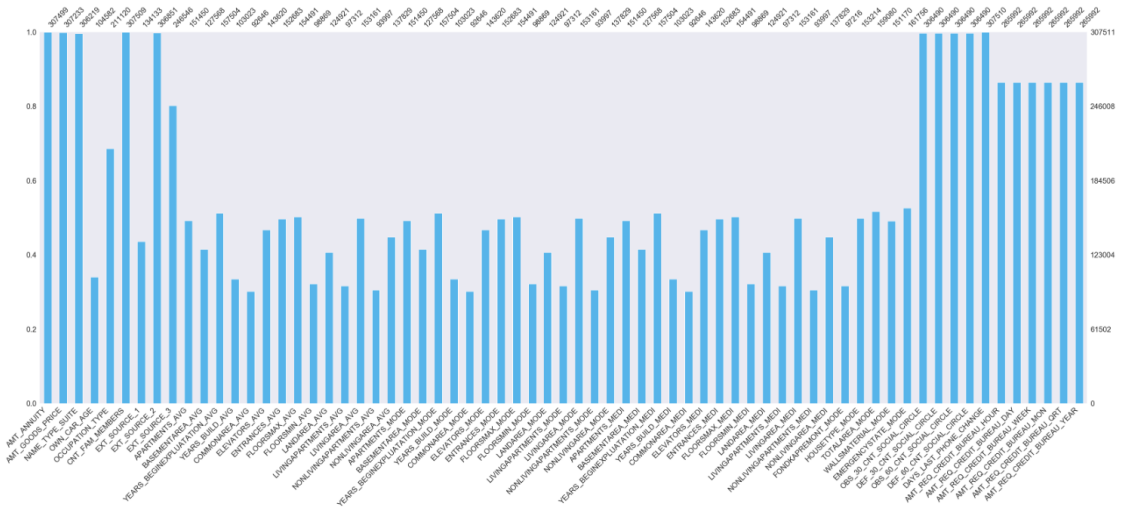

Figura 7. Valores não preenchidos

As características acima mostram o conjunto de dados como um todo. Durante a Análise Exploratória foi realizada também uma **análise individual** das distribuições de cada feature, de forma a identificar oportunidades de Engenharia de Features. Foram realizadas análises individuais para as 7 features mais preditivas, de acordo com o modelo baseline.

Visualização Exploratória

Uma visualização importante para o desenvolvimento do projeto foi o gráfico da Figura 8. Nele é possível ver as features mais importantes de acordo com o modelo **XGBoostClassifier** utilizado de forma inicial, sem tratamento especial de pré-processamento, que será explicado nas seções seguintes. O eixo vertical fornece o nome da feature, e o eixo horizontal fornece o score de importância da feature de acordo com o atributo **feature_importances_** do modelo.

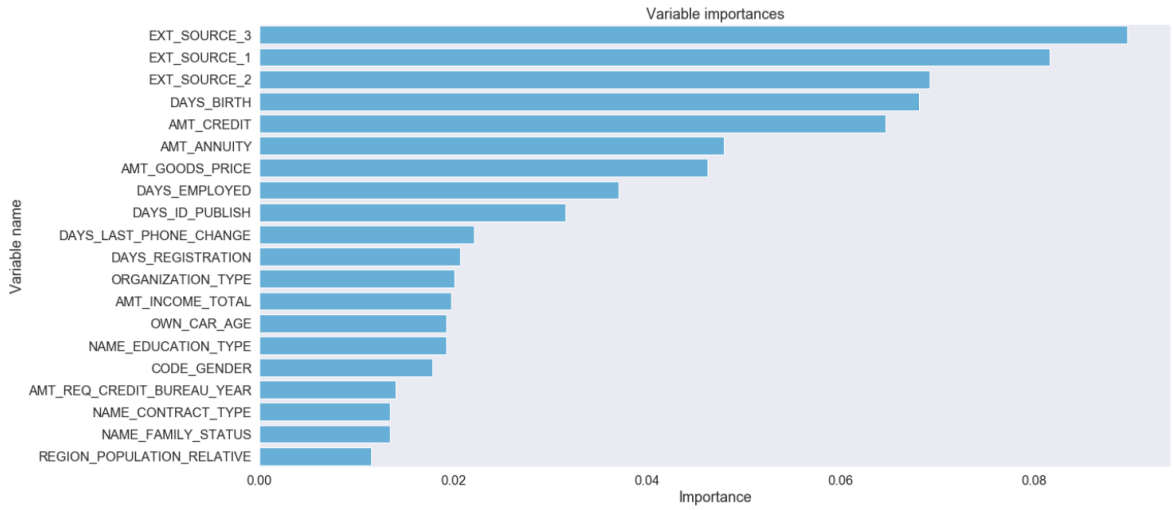

Figura 8. Features mais importantes - análise inicial.

Dentre todas as features do conjunto de dados, estas são as que possuem maior poder para diferenciar o cliente em relação ao target.

Algoritmos e Técnicas

Foram considerados modelos baseados em árvore e modelos lineares para comparação do poder preditivo, com relação aos dados utilizados. Seguem os modelos:

- Biblioteca XGBoost – **XGBClassifier**: este é um modelo bastante robusto, que vem sendo utilizado em soluções vencedoras de diversas competições de Ciência de Dados. É classificado como sendo um método de agregação, dentro da técnica **Gradient Boosted Trees** (no caso do parâmetro booster='gbtree'). Esta técnica cria árvores de decisão através de iterações de forma a fortalecer o poder preditivo da árvore anterior, utilizando gradientes da função objetivo definida.

  As árvores de decisão possuem a característica de **alto grau de variância** e **baixo grau de viés**. Neste modelo a variância é controlada utilizando-se parâmetros como **número de iterações**, **máxima profundidade** da árvore, **regularização**, etc. Com relação ao viés é possível utilizar parâmetros para uso de apenas **parte das observações** e **parte das features** dos dados. Esse controle de parâmetros permite ao XGBClassifier controlar o balanço entre viés e variância.

- Biblioteca Sklearn – **RandomForestClassifier**: também é um método de agregação bastante robusto e muito utilizado. Neste caso é classificado dentro da técnica **Bagging**, que cria várias árvores de decisão “fracas” (com pouca complexidade) e agrega todas as predições realizadas pelas arvores no final.
  Cada árvore de decisão possui um **alto grau de viés** e um **baixo grau de variância**, uma vez que possui baixa complexidade. Além disso, para cada árvore de decisão criada é utilizada apenas uma **parte das observações** e uma **parte das features** existentes, reforçando o alto grau de viés. Porém a predição final considera todas as árvores criadas, gerando assim uma predição com menor grau de viés e ainda o baixo grau de variância existe em cada árvore criada. A ilustra o modelo.

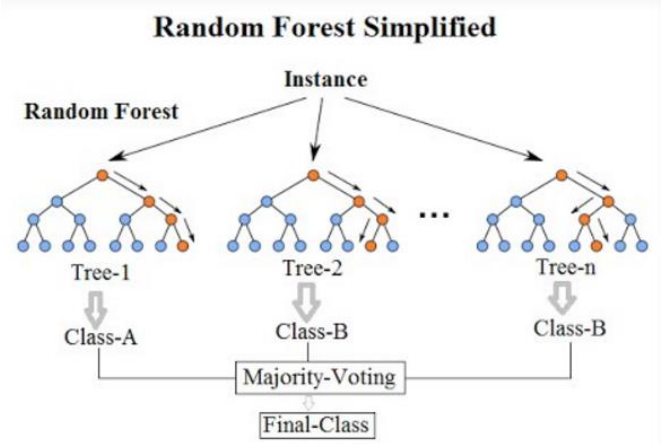

Figura 9. Modelo Random Forest.

Fonte: https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d

- Biblioteca Sklearn – **LogisticRegression**: se encontra na classe de classificadores lineares e possui sua saída diretamente associada a probabilidades. O objetivo do modelo é calcular os coeficientes **b** da Figura 10 de forma que a probabilidade **P** na saída dos dados explique de forma **otimizada** os valores target. A saída deste modelo se encontra entre os valores 0 e 1, pois representam probabilidades, conforme a Figura 11.

$$\ln\left(\frac{P}{1-P}\right) = b_0 + b_1 \times X_1 + b_2 \times X_2 \ldots\ldots + b_K \times X_K$$

Figura 10. Equação da Regressão Logística.

Fonte: https://techdifferences.com/difference-between-linear-and-logistic-regression.html

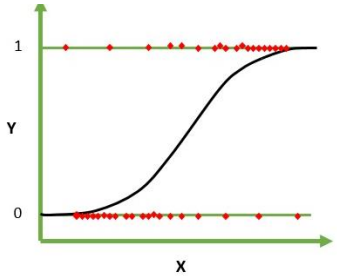

Figura 11. Saída da Regressão Logística.

Fonte: https://techdifferences.com/difference-between-linear-and-logistic-regression.html

Benchmark

Foi considerado como benchmark o modelo XGBClassifier com o pré-processamento básico das features, conforme a Figura 12.

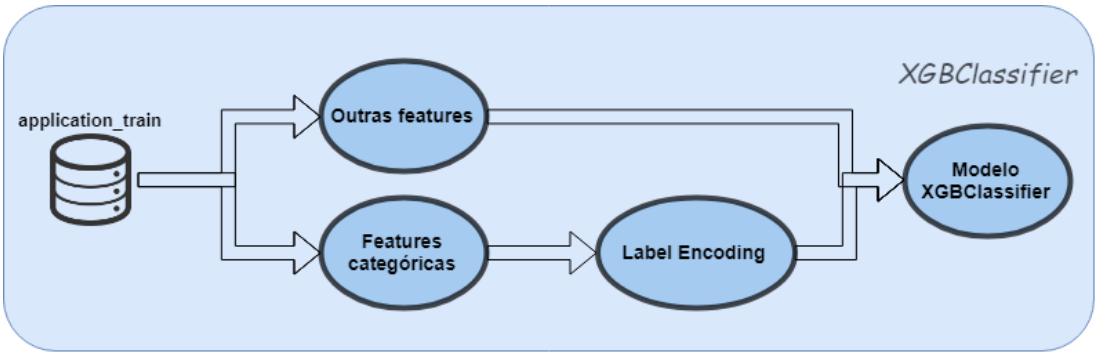

Figura 12. Modelo benchmark - XGBClassifier.

Foi realizado Label Encoding, em contraste com o One Hot Encoding, pois modelos baseados em árvore tendem a conseguir separar as features com esse tipo de técnica. Além disso o Label Encoding não aumenta a dimensionalidade do problema, em contraste com o One Hot Encoding que cria novas features.

Os outros modelos foram também testados, porém a sua performance foi inferior ao do XGBClassifier. Eles são descritos a seguir.

Como o RandomForestClassifier não possui tratativa dos números não preenchidos (NaNs) em seu algoritmo, foi necessário também inserir algum valor. Foi escolhido o número -999. A Figura 13 mostra o seu diagrama.

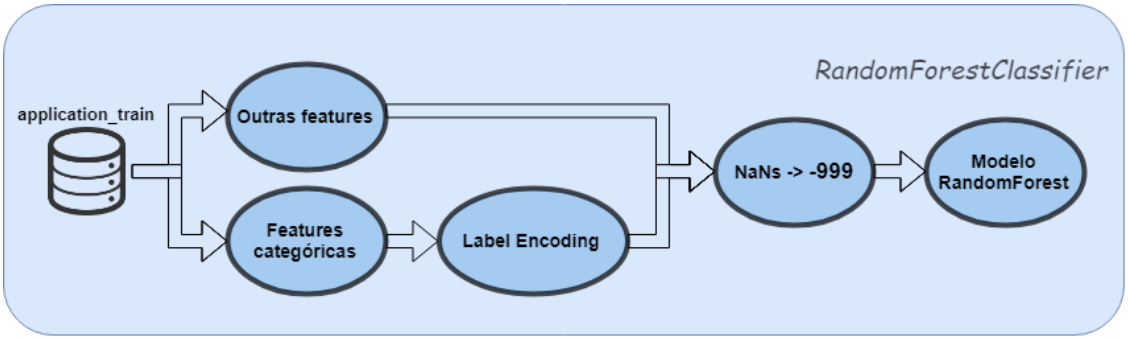

Figura 13. RandomForest com pré-processamento básico.

O One Hot Encoding tende a ser mais utilizado em modelos lineares, e foi utilizado para o modelo LogisticRegression conforme a Figura 14.

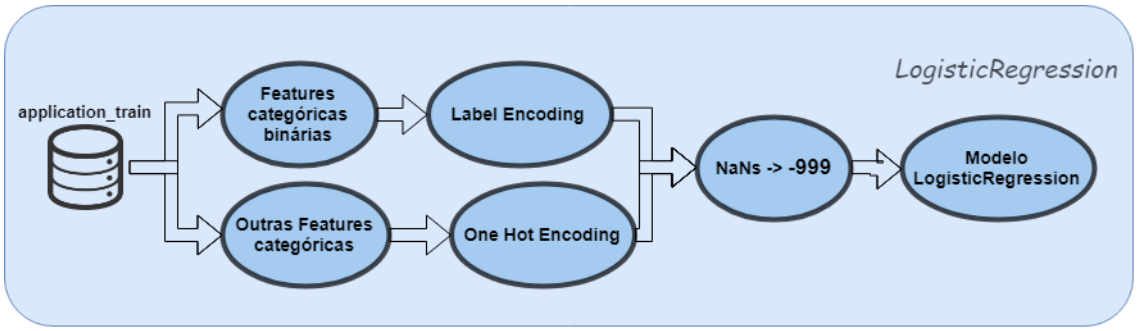

Figura 14. LogisticRegression com pré-processamento básico.

A performance dos modelos foi comparada utilizando todos os dados de treinamento e a função **cross_val_model** (arquivo evaluation.py). Foi utilizado o valor para o atributo **random_state = 10** nas funções que utilizam, de forma a gerarmos reprodutibilidade e podermos comparar os valores. Os valores médios encontrados foram os seguintes:

- XGBClassifier: **0.75624**
- RandomForestClassifier: **0.72245**
- LogisticRegression: **0.63599**

Com isso, foi escolhido o modelo XGBClassifier como sendo o modelo baseline. Foi realizada também um submissão à plataforma da Kaggle utilizando as predições de probabilidade de inadimplência deste modelo para os clientes do arquivo **application_test.csv**, de forma a termos mais uma forma de comparar a performance dos modelos subsequentes. Ao realizar a submissão a plataforma calcula a AUCROC com base no valor real do target (se foi inadimplente ou não) e as probabilidades fornecidas, gerando assim uma estimativa da performance em dados de fora da amostra. O valor da submissão é mostrado na Figura 15.

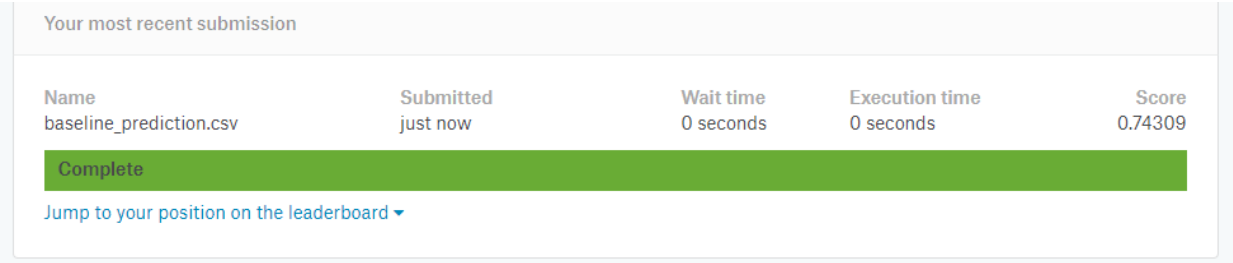

Figura 15. Submissão baseline na Kaggle.

Metodologia

Pré-Processamento de Dados

Primeiramente foi realizado um procedimento de **Seleção de Features** para reduzir a dimensionalidade do problema e desta forma melhorar a performance. Foram comparados modelos com o pré-processamento básico mostrado na Figura 12 porém com um número de features reduzido utilizando o atributo **features_importances_** do XGBClassifier. Desta forma foram escolhidas as **40 features** mais importantes, utilizando o seu poder preditivo, conforme a Figura 16.

Experiment:

Cross val XGB with only important features. Does the score goes up?

Results:

6 important features:

```
print("Cross_scores: ", cross_score)
print("Mean cross_score: %.5f" % cross_score.mean())
```

```
Cross_scores:  [0.73901287 0.73339048 0.74111273]
Mean cross_score: 0.73784
```

10 important features:

```
print("Cross_scores: ", cross_score)
print("Mean cross_score: %.5f" % cross_score.mean())
```

```
Cross_scores:  [0.74583663 0.74368159 0.75034866]
Mean cross_score: 0.74662
```

20 important features:

```
print("Cross_scores: ", cross_score)
print("Mean cross_score: %.5f" % cross_score.mean())
```

```
Cross_scores:  [0.75545365 0.75254123 0.7591305 ]
Mean cross_score: 0.75571
```

40 important features:

```
print("Cross_scores: ", cross_score)
print("Mean cross_score: %.5f" % cross_score.mean())
```

```
Cross_scores:  [0.75928768 0.75352727 0.7592095 ]
Mean cross score: 0.75734
```

60 important features:

```
print("Cross_scores: ", cross_score)
print("Mean cross_score: %.5f" % cross_score.mean())
```

```
Cross_scores:  [0.75686671 0.75221049 0.75928009]
Mean cross_score: 0.75612
```

80 important features:

```
print("Cross_scores: ", cross_score)
print("Mean cross_score: %.5f" % cross_score.mean())
```

```
Cross_scores:  [0.75700251 0.75256538 0.75916569]
Mean cross_score: 0.75624
```

102 important features (that have a positive importance value):

```
print("Cross_scores: ", cross_score)
print("Mean cross_score: %.5f" % cross_score.mean())
```

```
Cross_scores:  [0.7571421  0.75299974 0.75937974]
Mean cross_score: 0.75651
```

Figura 16. Experimento - Redução de Dimensionalidade + Label Encoding. Poder preditivo dos modelos.

Após isso foram realizados os seguintes procedimentos de Engenharia de Features com base na exploração dos dados:

- Criação de uma coluna **"FEATURE_NAN"**, de forma a identificar se a feature em questão possui um valor NaN.
- Criação de uma coluna **"FEATURE_IS_ANOM"**, de forma a identificar se a feature em questão é um outlier (de acordo com o método de Tukey para detecção de outliers).

Os resultados com as melhores features criadas são mostrados na Figura 17.

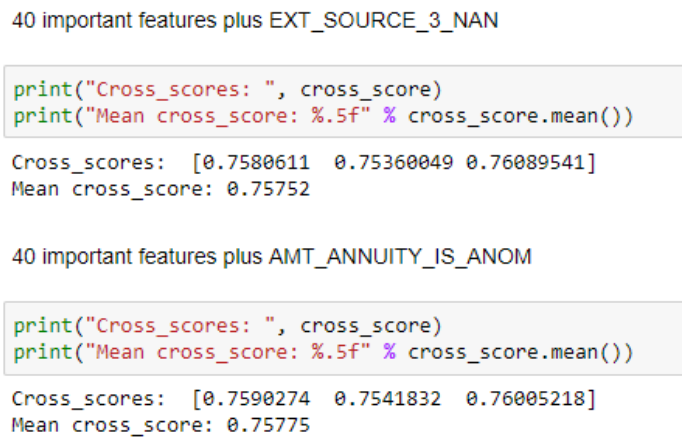

Figura 17. Feature Engineering - Resultados.

Com relação à substituição dos valores não preenchidos, não é necessário para o modelo XGBClassifier. Este é um algoritmo que possui uma tratativa para estes casos. Ele cria um caminho padrão para as separações das árvores de decisão criadas durante o algoritmo, de forma que se um exemplo não possui um valor para uma feature específica, ele é encaixado na separação padrão. Mais informações no artigo "**XGBoost: A Scalable Tree Boosting System**".

Implementação

A implementação inicial se baseou no seguinte fluxo resumido:

- **Pré-processamento**
  - Label Encoding.
- **Seleção de Features**
  - features_importances_ do XGBClassifier.
- **Extração de Features**
  - Criação de features **"FEATURE_NAN"** e **"FEATURE_IS_ANOM".**
- **Modelo**
  - Utilização do XGBClassifier com parâmetros iniciais razoáveis, considerando sobreajuste.
- **Refinamento de parâmetros**
  - Utilização de validação cruzada aninhada para identificação de parâmetros ideais e estimar o valor da AUCROC fora do conjunto de treino.

Com relação aos parâmetros iniciais do modelo, foram escolhidos os seguintes:

- **learning_rate = 0.1**
    - Este é um parâmetro que altera de forma significativa o comportamento do modelo. Foi fixado primeiramente para podermos refinar os outros parâmetros. Após o refinamento é possível escolher um novo valor para learning_rate e realizar um novo ciclo de refinamento.
- **n_estimators = 1000**
    - O valor 1000 foi escolhido primeiramente antes do refinamento. Este é um valor alto, que pode causar sobreajuste dos dados. Porém foi utilizada a função de validação cruzada nativa do pacote xgboost juntamente com o parâmetro **early_stopping_rounds**. Com isso foi possível refinar este parâmetro com base nos outros parâmetros do modelo ao longo do refinamento.
- **objective = 'rank:pairwise'**
    - Este valor foi escolhido para o modelo trabalhar em cima do problema de rankeamento explicado na seção **Métricas** do presente trabalho. Isto fez com que o algoritmo fosse capaz de otimizar a ordem dos scores de probabilidade a cada **par** de exemplos do conjunto de dados.
- **n_jobs = 4**
    - O código foi utilizado em máquina local, e este parâmetro permitiu a utilização da CPU do computador por completa.
- **max_depth = 5**
    - Controla o sobreajuste aos dados.
- **min_child_weight = 1**
    - Controla o sobreajuste aos dados.
- **gamma = 0**
    - Controla o sobreajuste aos dados.
- **subsample = 0.8**
    - Controla o sobreajuste aos dados.
- **colsample_bytree = 0.8**
    - Controla o sobreajuste aos dados.
- **reg_alpha = 0**
    - Valor padrão. Regularização L1.
- **reg_lambda = 1**
    - Valor padrão. Regularização L2.
- **scale_pos_weight = 1**
    - Útil para problemas com classes desbalanceadas. Foi escolhido inicialmente o valor padrão.
- **random_state = 10**
    - Utilizado para reprodutibilidade e comparação dos resultados.

Refinamento

A Figura 18 representa o fluxo utilizado para refinamento dos parâmetros do modelo XGBClassifier. Este fluxo é muito bem explicado por Aarshay Jain no blog **Analytics Vidhya**.

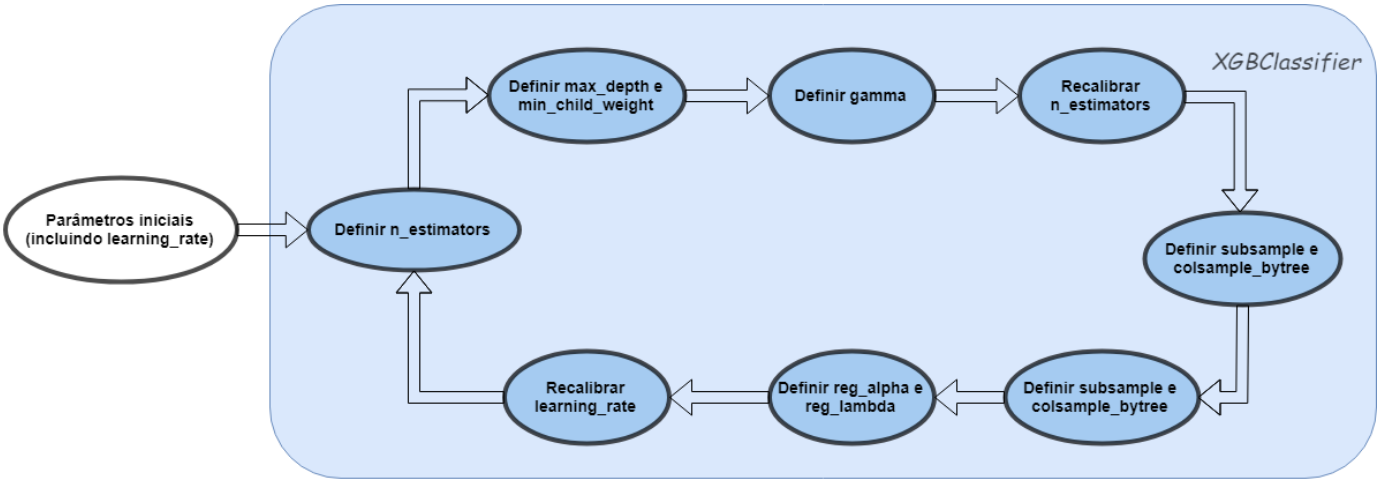

Figura 18. Refinamento dos parâmetros - XGBClassifier.

Foi realizado apenas um ciclo de refinamento no presente trabalho, devido ao tempo necessário para a máquina local processar. Os valores encontrados foram:

- **learning_rate = 0.1**
- **n_estimators = 616**
- **max_depth = 4**
- **min_child_weight = 9**
- **gamma = 0.05**
- **subsample = 0.75**
- **colsample_bytree = 0.85**

Resultados

Modelo de Avaliação e Validação

Com os parâmetros otimizados encontrados na seção "**Refinamento**" foi treinado um novo modelo XGBClassifier e avaliado com base nos dados de teste separados no início do refinamento. A Figura 19 mostra o resultado.

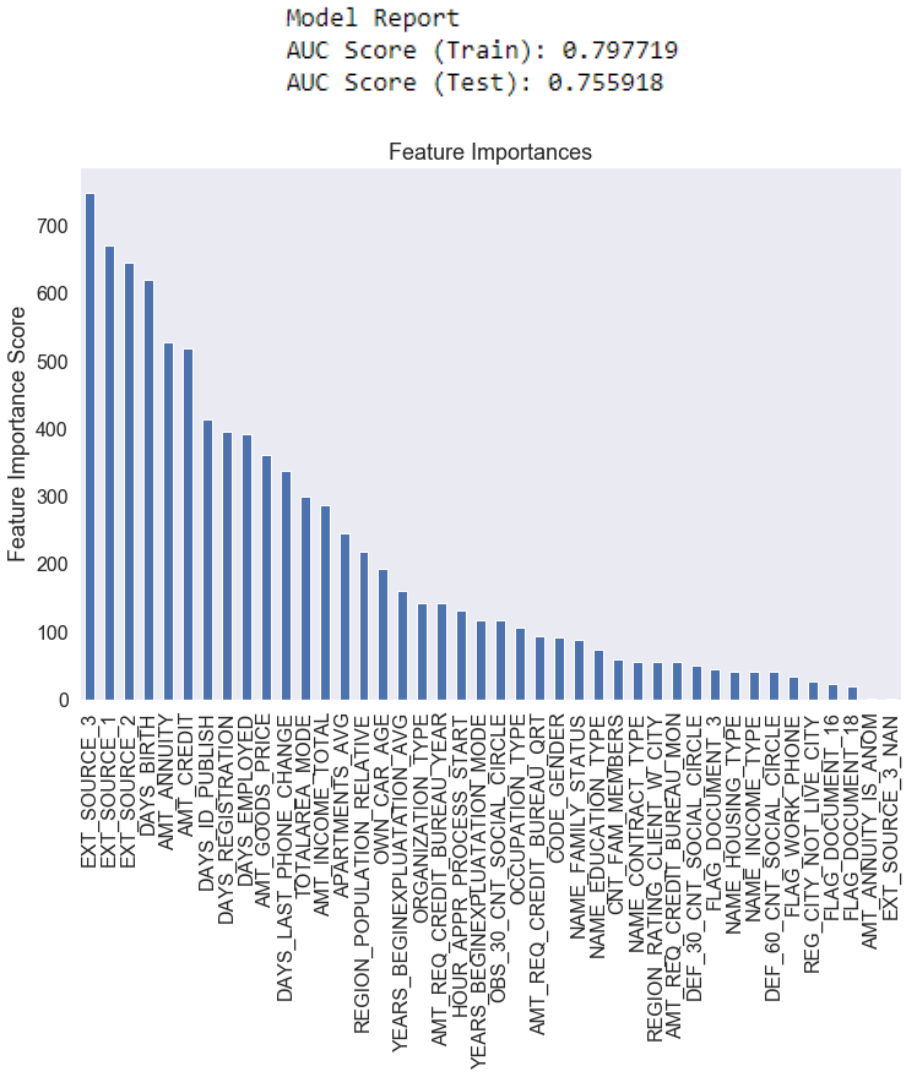

Figura 19. Modelo final.

Com isso foi possível obter uma estimativa final da performance fora da amostra que os parâmetros otimizados podem nos fornecer.

Finalmente foi treinado um modelo com todos os dados do arquivo **application_train.csv**. O objetivo deste modelo é fazer as predições de risco de inadimplência para o conjunto **application_test.csv** e realizar a submissão na Kaggle. Portanto ainda foi utilizado o parâmetro **random_state**, de forma a comparar com o modelo baseline.

É importante ressaltar que foi utilizada a técnica de validação cruzada aninhada durante o refinamento do modelo. Isso permitiu as seguintes características com relação à robustez do modelo:

- O ciclo interno da validação cruzada permitiu a escolha de parâmetros otimizados em cada conjunto do ciclo externo.
- O ciclo externo permitiu gerarmos mais de uma amostra do conjunto de parâmetros otimizados, e ao mesmo tempo termos uma estimativa da performance fora da amostra.
- Minimizar o vazamento de informação durante as validações.

Justificativa

Para a comparação com o modelo básico mostrado na seção **Benchmark** foi utilizada a função **cross_val_model** (arquivo evaluation.py) e uma submissão das predições de probabilidade de inadimplência dos clientes do arquivo **application_test.csv**. Os valores são mostrados na Tabela 1, juntamente com os valores do modelo baseline para comparação. A Figura 20 mostra a submissão na Kaggle.

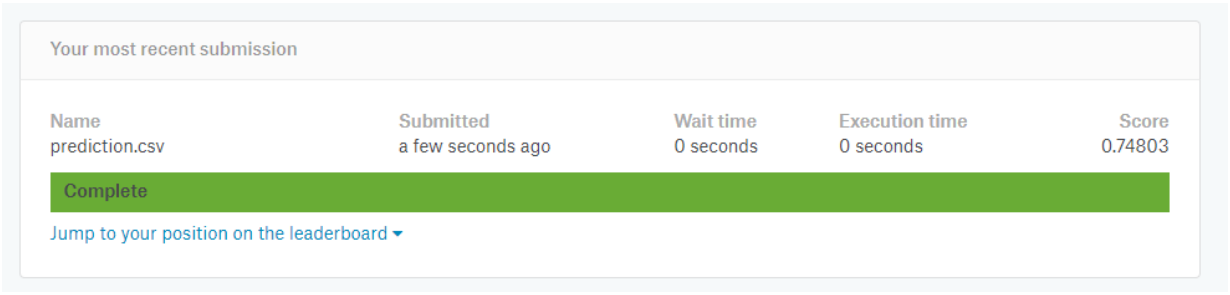

Figura 20. Submissão do modelo final na Kaggle.

Tabela 1. Resultados do modelo final e comparação.

Modelo	Média cross_val_model	Submissão
Baseline	0.75624	0.74309
Final	0.75893	0.74803
Aumento:	0.36%	0.66%

Conclusão

Forma Livre de Visualização

Conforme explicado na seção "**Métricas**", o problema exposto no presente projeto se encaixa em um problema de **rankeamento**. Nesse contexto o objetivo é o modelo final obter um **alto grau de separabilidade** entre as probabilidades das 2 classes: cliente inadimplente e cliente não inadimplente. Para isso foi utilizado como objetivo do modelo XGBClassifier o parâmetro "**rank:pairwise**", de forma a penalizar um alto número de pares de pontos não ordenados (rankeados).

A Figura 21 mostra o grau de separabilidade alcançado pelo modelo final, através de um histograma.

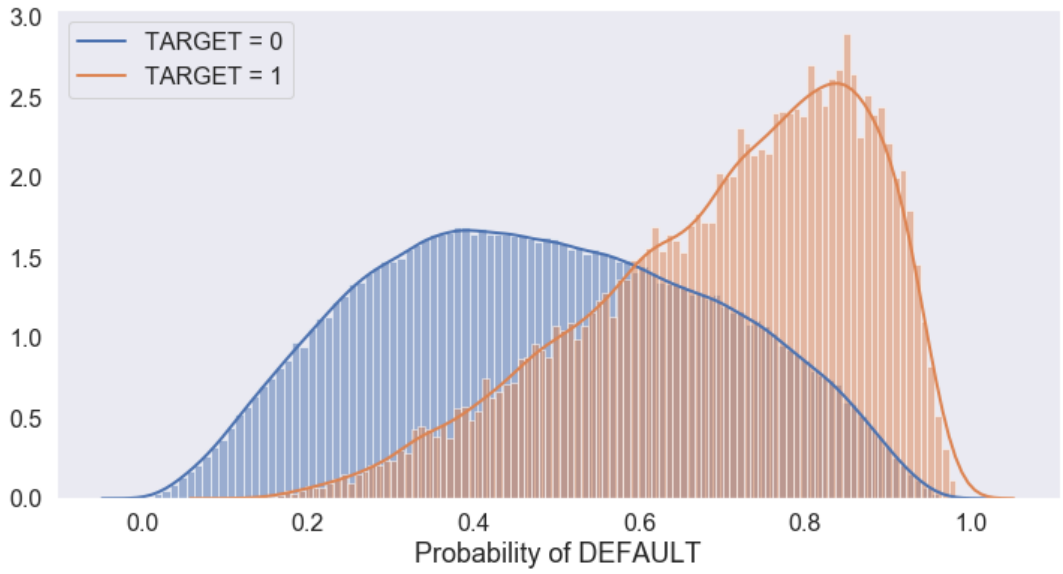

Figura 21. Ilustração da separabilidade entre as classes.

Reflexão

O seguinte fluxo resumido foi utilizado para desenvolvimento do projeto:

- Análise geral do conjunto de dados considerado (application_train.csv);
- Criação de um modelo baseline com pré-processamento básico;
- Análise específica das features mais preditivas, de acordo com o modelo baseline;
- Engenharia de Features, com base nas análises feitas;
- Refinamento de parâmetros do modelo;
- Validação e comparação dos resultados.

O projeto foi iniciado com a premissa de construir uma análise inicial e básica do problema, de forma que ao longo do tempo e de outras implementações o poder preditivo seja melhorado. Essa é uma premissa válida na área de Ciência de Dados, que ajuda no andamento e evolução gradual das análises. Com base nisso foi feita a simplificação inicial de utilizar apenas o arquivo **application_train.csv**.

Um desafio identificado durante o projeto foi o número de features. Em apenas um arquivo, temos 121 features, o que fez com que a análise individual seja demorada. Devido ao tempo do projeto não foi possível realizar a análise de todas as features.

Com relação à usabilidade geral do modelo **não é possível generalizar** o poder preditivo deste modelo construído, de forma a ser usado em outros problemas. Cada problema possui o seu **perfil de clientes**, o que faz com que deva ser validado um modelo específico para cada situação. Ainda, mesmo em condições específicas de um problema, é necessária a **monitoração dos dados e suas distribuições** enquanto o modelo está em produção, pois o perfil dos clientes pode mudar ao longo do tempo.

Melhorias

Uma melhoria que deve ser feita em futuras implementações deste projeto é o exploração e utilização dos dados dos outros arquivos fornecidos pelos organizadores da competição. É possível que este conjunto de dados possua uma quantidade de informação que não foi utilizada na presente implementação.

Novas implementações devem considerar também a exploração das outras features existentes no arquivo **application_train.csv**, para identificação de oportunidades específicas de Engenharia de Features.

Deve ser considerado também o estudo do domínio do problema (fornecimento de crédito) para identificação de oportunidades de criação de novas features informativas, de acordo com conhecimento do negócio. É possível que na literatura exista o direcionamento para features ou técnicas com alto poder preditivo.

Outra abordagem que pode ser considerada é a redução de dimensionalidade do problema utilizando-se técnicas de aprendizagem não-supervisionada, como PCA por exemplo. Com isso, é possível testar o poder preditivo de novos modelos.

Para confecção de melhores modelos é possível também a utilização de técnicas de agregação para mais de 2 modelos, de forma a potencializar a performance.

As oportunidades aqui descritas não foram utilizadas devido ao tempo restante no curso.

Referências

"Home Credit Default Risk" - https://www.kaggle.com/c/home-credit-default-risk

"Credit risk measurement: Developments over the last 20 years", Altman, Saunders (1998) - http://socsci2.ucsd.edu/~aronatas/project/academic/science.pdf

"**Estudo Comparativo entre Técnicas de Aprendizado de Máquina para Estimação de Risco de Crédito"**, Aniceto (2016) - http://repositorio.unb.br/handle/10482/20522

"Understanding AUC - ROC Curve" - https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

"XGBoost: A Scalable Tree Boosting System" - https://arxiv.org/abs/1603.02754

"Complete Guide to Parameter Tuning in XGBoost (with codes in Python)" - https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

"Random Forest Simple Explanation" - https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d

"Difference Between Linear and Logistic Regression" - https://techdifferences.com/difference-between-linear-and-logistic-regression.html