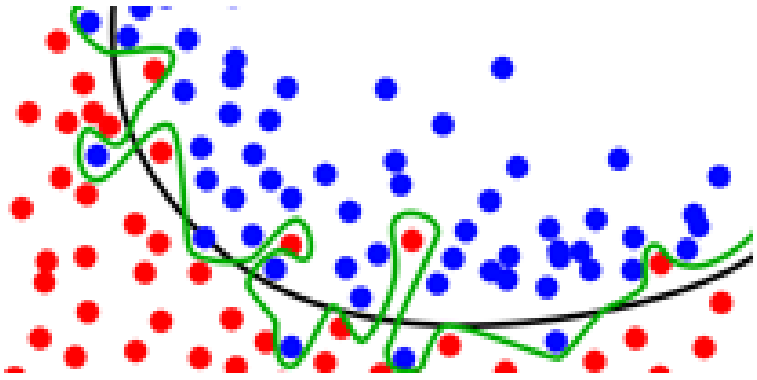
Capstone Project versão 1.0
Don´t Overfit!

Curso: Machine Learning Engineer Nanodegree



Ademar Silva Barreto Junior
26/5/2018

Definição

Visão Geral do Projeto

O objetivo do trabalho é fazer um análise do desafio escolhido do kaggle (Don´t Overfit!) de 2011.

O acesso no Kaggle a este desafio é :

https://www.kaggle.com/c/overfitting

Este desafio visa estimular a pesquisa e destacar os algoritmos, técnicas ou estratégias existentes que podem ser usados para se proteger contra o overfitting .

A pergunta feita é quais são as melhores técnicas para evitar o overfitting?

Um passo necessário na construção de modelos é garantir que eles não provoquem Overfit com dados de treinamento, o que leva a previsões sub-otimizadas em novos dados.

Para conseguir isso, foi criado um conjunto de dados simulado com 200 variáveis e 20.000 casos.

Uma "equação" baseada nesses dados foi criada para gerar um Destino a ser previsto.

Dado os 20.000 casos, o problema é muito fácil de resolver - mas você recebe apenas o valor alvo de 250 casos - a tarefa é construir um modelo que ofereça as melhores previsões nos restantes 19.750 casos.

Esta competição é particularmente relevante para análise de dados médicos, onde muitas vezes o número de casos é severamente restringido.

Declaração do problema

O problema a ser resolvido é qual o modelo de algoritmo que melhor resolve o problema de overfitting.

Como evitar o overfitting?

Overfitting refere-se a um modelo análise os dados de treinamento muito bem.

O overfitting ocorre quando um modelo aprende os detalhes e o ruído nos dados de treinamento, na medida em que afeta negativamente o desempenho do modelo em novos dados.

Isso significa que o ruído ou as flutuações aleatórias nos dados de treinamento são captados e aprendidos como conceitos pelo modelo.

O problema é que esses conceitos não se aplicam a novos dados e afetam negativamente a capacidade de generalização dos modelos.

O problema **Don´t Overfit!** Proposto pelo Kaggle em 2011 é um problema de classificação.

Para analisar a base de dados fornecida em 2018, utilizamos algoritmos de Machine Learning mais recentes e que teoricamente melhorará as nossas conclusões.

Os algoritmos de aprendizagem baseados em árvores são considerados um dos melhores e mais utilizados métodos de aprendizagem supervisionada. Os métodos baseados em árvore habilitam modelos preditivos com alta precisão, estabilidade e facilidade de interpretação. Ao contrário dos modelos lineares, eles mapeiam as relações não-lineares muito bem. Eles são adaptáveis para resolver qualquer tipo de problema em questão (classificação ou regressão).

Métodos como árvores de decisão, floresta aleatória, aumento gradiente estão sendo popularmente usados em todos os tipos de problemas de Data Science.

Algoritmos utilizados

Os mais algoritmos classificadores escolhidos para análise foram os seguintes:

LightGBM (LGBMClassifier)

Como ele difere de outro algoritmo baseado em árvore?

O LightGBM leve cresce verticalmente enquanto outros algoritmos crecem árvores na horizontal, o que significa que o
GBM Light cresce em termos de folha de árvore enquanto outro algoritmo cresce no nível.

Por que o Light GBM está ganhando popularidade extrema?

O tamanho dos dados está aumentando dia a dia e está se tornando difícil para os algoritmos tradicionais da ciência de dados obter resultados mais rápidos. O Light GBM é considerado como "Light" por causa de sua alta velocidade. O Light GBM pode manipular o tamanho grande de dados e leva menos memória para ser executado. Outra razão pela qual o Light GBM é popular é porque se concentra na precisão dos resultados. O LGBM também suporta o aprendizado de GPU e, portanto, os cientistas de dados estão usando amplamente o LGBM para o desenvolvimento de aplicativos de ciência de dados.

O Light GBM pode ser utilizado em todos os casos?
Não é aconselhável usar o LGBM em pequenos conjuntos de dados. O Light GBM leve é sensível ao overfitting e pode facilmente ajustar pequenos dados. Não há limite para o número de linhas, mas minha experiência sugere que eu use apenas para dados com mais de 10.000 linhas que é o caso da base de dados que tem 20000 linhas.

CatBoost (CatBoostClassifier)

O CatBoost fornece um bom recurso para evitar overfitting. Se você definir iterações para serem altas, o classificador irá usar muitas árvores para construir o classificador final e você corre o risco de overfitting.

Random Forest (RandomForestClassifier)

Random Forest superam vários problemas com árvores de decisão, incluindo:

- Redução no overfitting: pela média de várias árvores, há um risco significativamente menor de overfitting.

- Menos variação: ao usar várias árvores, você reduz a chance de tropeçar em um classificador que não apresenta bom desempenho devido à relação entre os dados de treinamento e d teste.

Como consequência, em quase todos os casos, as Random Forest são mais precisas do que as árvores de decisão.

Xgboost (XGBClassifier)

XGBoost (eXtreme Gradient Boosting) é um algoritmo que tem dominado recentemente a aprendizagem de máquina aplicada nas competições recentes do Kaggle para dados estruturados ou tabulares.

O XGBoost é uma implementação de gradient boosting de árvores focada e projetada para velocidade e desempenho.

As duas razões para usar o XGBoost são portanto:

- Velocidade de Execução.
- Desempenho do Modelo.

Métricas de avaliação

Uma especificação exata do algoritmo de treinamento utilizado é fundamental. Ao utilizar um algoritmo conhecido, especifique com referências que descrevam e utilizam o algoritmo de maneira precisa.

Este é o procedimento que adotamos em nossa pesquisa (validação cruzada) e Algoritmos de Treinamento.

Para a validação cruzada (cross validation) dos dados utilizei "StratifiedK-Fold do Scikit Learn".

A definição das classes é a seguinte:

- from sklearn.model_selection import StratifiedKFold
- from sklearn.model_selection import cross_val_score

A validação cruzada (cross validation) é uma técnica para avaliar a capacidade de generalização de um modelo, a partir de um conjunto de dados. Esta técnica é amplamente empregada em problemas onde o objetivo da modelagem é a predição.

Busca-se então estimar o quão preciso é este modelo na prática, ou seja, o seu desempenho para um novo conjunto de dados.

A ideia é dividir a massa de dados disponível em k partições (os tais "*folds*") e realizar k rodadas de treinamento e teste com essas combinações de dados. Minimiza-se desta forma as chances de algum dado importante para a classificação ser deixado de fora durante o treinamento, pois temos 200 variáveis nesta base de dados.

O "Stratified K-Fold" garante que sempre haverá um percentual equivalente de dados

A descrição dos parâmetros que utilizaremos fundamentar a nossa análise serão:

Receiver Operating Characteristic Area Under the Curve (ROC AUC)

É uma medida do desempenho do classificador, que é amplamente usado no aprendizado de máquina.

Infelizmente, a forma obscura como o conceito é explicado na maioria das fontes dificulta a compreensão do seu significado intuitivo.

O nome "área sob a curva" é mal concebido e é totalmente inútil para ajudar a intuição.

O objetivo desta explicação é ajudar aqueles que lutam com o conceito e apresentar uma interpretação simples e intuitiva da métrica ROC AUC como a **"classificação positiva média"** que não vi declarado explicitamente em outro lugar.

Acurácia

A primeira métrica a se utilizar é a acurácia de classificação. Mas ela não é suficiente para decidirmos qual o melhor classificador. Por isso utilizaremos a ROC AUC.

As métricas de avaliação serão feitas pelo cálculo ROC AUC e ACURÁCIA e gráficos feitos seaborn para verificar como o resultado de o e1 estarão separados e não intercalados.

Quanto maior o ROC AUC e a ACURÁCIA, melhor será o nosso modelo. Quanto maior o ROC AUC e ACURÁCIA dos dados teste em relação aos dados de teste, melhor será o algoritmo e será este que deveríamos submeter ao Kaggle.

Descrição da Base de Dados

O arquivo de dados contém 200 variáveis geradas aleatoriamente, var_1 a var_200.

Existem 20.000 linhas de dados, das quais você recebe apenas o 'Target' para os primeiros 250. O 'Target' é 1 ou 0, então este é um problema de classificação.

Existem também 5 outros campos:

- case_id - 1 a 20.000, um identificador exclusivo para cada linha
- train - 1/0, este é um sinalizador para as primeiras 250 linhas que são o conjunto de dados de treinamento
- Target_Practice - fornecemos todas as 20.000 metas para esse modelo, para que você possa desenvolver seu método completamente off-line.
- Target_Leaderboard - apenas 250 alvos são fornecidos. Deve ser enviado as previsões para os 19.750 restantes na tabela de classificação de Kaggle.
- Target_Evaluate - novamente, apenas 250 alvos são fornecidos. Os competidores que vencerem o 'benchmark' na Tabela de Líderes serão solicitados a fazer uma submissão adicional para o modelo de Avaliação.

As previsões enviadas devem conter duas colunas, separadas por uma vírgula, com uma linha de cabeçalho. A primeira coluna é o case_id e a segunda coluna é a previsão para o Target_Leaderboard.

O case_id deve ser de 251 a 20.000, em ordem crescente.

A previsão pode ser qualquer número real. Este é um desafio de ordem de classificação, então não está restrito a apenas 1s ou 0s.

A estrutura da base de dados é a seguinte:

CASE_ID	TRAIN	TARGET_PRACTICE	TARGET_LEADERBOARD	TARGET_EVALUATE	VAR_1 A VAR_200

Descrição da solução e seus resultados

Inicialmente será feita uma **análise dos dados** para verificação de missings (valores nulos nas tabelas que precisam ser tratados antes de submetê-los aos algoritmos), tabela de correlação das variáveis e verificação das variáveis mais significativas.

Na análise de dados, temos os seguintes passos:

Inicialização das bibliotecas

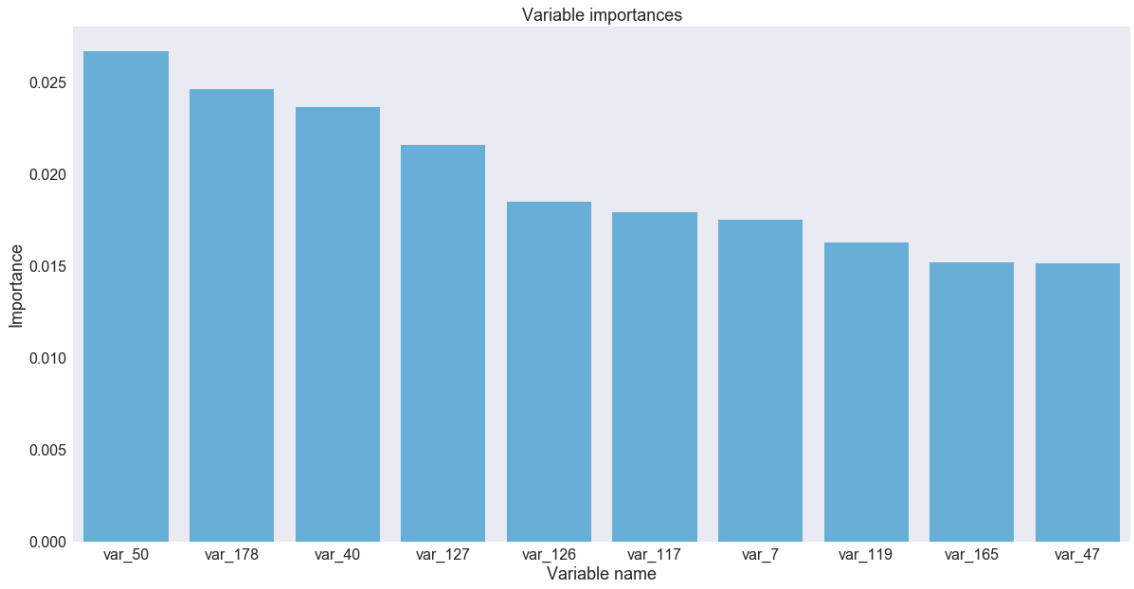
Leitura da Base de Dados e divisão da mesma em dados de treinamento e teste

Análise dos dados classificando em target, input, nominal e interval

Verificação dos valores de missings

Tabela de correlação entre as variáveis

Verificação das feature importance onde obtivemos o seguinte gráfico:



Na solução do problema, serão utilizados 4 algoritmos individualmente, além de utilizar técnicas de ensemble e one-hot enconde para calcular o ROC_AUC e acurácia e verificar qual deles tem melhor performance para evitar o overfitting.

O primeiro passo foi fazer uma análise da Base de Dados utilizando os algoritmos individualmente. Esta etapa chamamos de Modelos de Baseline que são os seguintes:

Resultado Modelos de Baseline¶

Modelos de Baseline¶

Nesta etapa utilizamos cross validation e depois aplicamos o modelo obtendo os seguintes resultados:

Protocolo LGBMClassifier	ROC_AUC Treinamento	ROC_AUC Teste	Acurácia Treinamento	Acurácia Teste
Fold 1	0.51325	0.57570	0.54242	0.51318
Fold 2	0.55446	0.45470	0.52403	0.53406
Fold 3	0.60728	0.47302	0.52955	0.52381

Protocolo CatBoostClassifier	ROC_AUC Treinamento	ROC_AUC Teste	Acurácia Treinamento	Acurácia Teste
Fold 1	0.44351	0.61093	0.51158	0.59524
Fold 2	0.53841	0.53841	0.55379	0.51190
Fold 3	0.63580	0.46990	0.48810	0.47443

Protocolo RandomForestClassifier	ROC_AUC Treinamento	ROC_AUC Teste	Acurácia Treinamento	Acurácia Teste
Fold 1	0.58626	0.50000	0.52998	0.52382
Fold 2	0.58210	0.50000	0.55433	0.52382
Fold 3	0.74554	0.50000	0.57123	0.52425

Protocolo XGBClassifier	ROC_AUC Treinamento	ROC_AUC Teste	Acurácia Treinamento	Acurácia Teste
Fold 1	0.59934	0.73956	0.56613	0.64170
Fold 2	0.55600	0.53968	0.53593	0.50981
Fold 3	0.76554	0.45372	: 0.74431	0.45062

Conclusão Modelos de Baseline

Pela tabela de correlação de Pearson, todas as variáveis (var_1 a var_200) apresentam correlação baixa ou negativa, entre 0 a 0.3 positivo ou negativo, que indica uma correlação desprezível. Portanto, não podemos desprezar nenhuma variável no momento de fazermos as nossas análises dos algoritmos de machine learning. Pela análise acima, todos os protocolos apresentaram acurácia e ROC AUC também baixo, o que mostra uma performance não muito boa de todos estes algoritmos.¶

Modelos de One-Hot Enconde

Nesta etapa utilizamos cross validation, fazemos One-hot Enconde nas variáveis categóricas e depois aplicamos o modelo obtendo os seguintes resultados:

Resultado Modelos de Baseline¶

Protocolo LGBMClassifier	ROC_AUC Treinamento	ROC_AUC Teste	Acurácia Treinamento	Acurácia Teste
Fold 1	0.51325	0.57570	0.54242	0.51318
Fold 2	0.55446	0.45470	0.52403	0.53406
Fold 3	0.60728	0.47302	0.52955	0.52381

Protocolo CatBoostClassifier	ROC_AUC Treinamento	ROC_AUC Teste	Acurácia Treinamento	Acurácia Teste
Fold 1	0.48850	0.57267	0.57186	0.60714
Fold 2	0.57887	0.45886	0.42760	0.55952
Fold 3	0.54966	0.54256	0.51786	0.54630

Protocolo RandomForestClassifier	ROC_AUC Treinamento	ROC_AUC Teste	Acurácia Treinamento	Acurácia Teste
Fold 1	0.58626	0.50000	0.52998	0.52382
Fold 2	0.58210	0.50000	0.55433	0.52382
Fold 3	0.74554	0.50000	0.57123	0.52425

Protocolo XGBClassifier	ROC_AUC Treinamento	ROC_AUC Teste	Acurácia Treinamento	Acurácia Teste
Fold 1	0.59934	0.73956	0.56613	0.64170
Fold 2	0.55600	0.53968	0.53593	0.50981
Fold 3	0.76554	0.45372	: 0.74431	0.45062

Conclusão One-Hot Enconde

Para esta base de dados que não possui dados categóricos e em nada adiantou fazer o One-Hot Enconde. Obtivemos os mesmos resultados dos Modelos de Baseline. Neste caso mostrou-se totalmente ineficaz.¶

Testes dos algoritmos sem stacking

Neste caso estamos com uma base de dados ampliada .Temos agora uma base de 200 colunas para uma de 600 colunas (200 colunas var_1 a var_200, 200 colunas var_1_mean_range a var_200_mean_range e 00 colunas var_1_median_range a var_200_median_range) e mais 4 colunas(train_stack['null_sum'], train_stack['bin_sum'] , train_stack['ord_sum'] e train_stack['interval_median'] o que dá um total de 604 colunas).¶

Resultado dos algoritmos sem stacking

Protocolo LGBMClassifier	ROC_AUC Treinamento	ROC_AUC Teste	Acurácia Treinamento	Acurácia Teste
Fold 1	0.69599	0.74750	0.63279	0.66431
Fold 2	0.73915	0.60952	0.63831	0.54893
Fold 3	0.69965	0.70525	0.58983	0.57231

Protocolo CatBoostClassifier	ROC_AUC Treinamento	ROC_AUC Teste	Acurácia Treinamento	Acurácia Teste
Fold 1	0.57480	0.56199	0.60855	0.45238
Fold 2	0.64884	0.51797	0.55433	0.55952
Fold 3	0.57512	0.67187	0.55952	0.52425

Protocolo RandomForestClassifier	ROC_AUC Treinamento	ROC_AUC Teste	Acurácia Treinamento	Acurácia Teste
Fold 1	0.65416	0.50000	0.54232	0.52382
Fold 2	0.67579	0.50000	0.53030	0.52382
Fold 3	0.70767	0.50000	0.58941	0 .52425

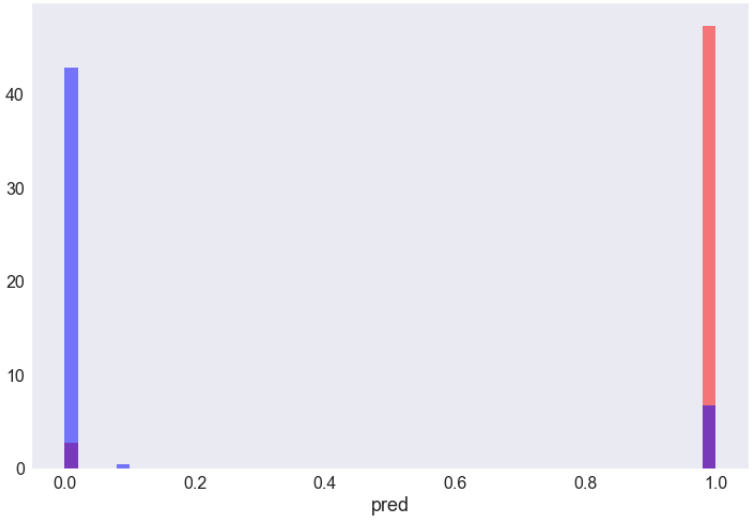
Protocolo XGBClassifier	ROC_AUC Treinamento	ROC_AUC Teste	Acurácia Treinamento	Acurácia Teste
Fold 1	0.71798	0.82222	0.63279	0.76213
Fold 2	0.78596	0.64542	0.68680	0.62983
Fold 3	0.81808	0.71331	0.74388	0.67063

CONCLUSÃO teste dos algoritmos sem stacking.

Os resultados foram bem superiores, principalmente para os algoritmos XGBClassifier (que achei o melhor) para evitar overfitting neste caso, e o LGBMClassifier que também apresentou ROC AUX e acurácia mais alta que o CatBoostClassifier e o RandomForestClassifier. Pelos gráficos que geramos acima, os protocolos XGBClassifier e ROC AUX, fazem uma separação mais eficiente de 0 e 1, o que mostra menos overfitting.

Pelo gráfico que geramos abaixo, os protocolos XGBClassifier e LGBMClassifier, fazem uma separação mais eficiente de 0 e 1, o que mostra menos overfitting.¶

LGBMClassifier – Separação de 0 e 1



Ensemble¶

O objetivo do ensemble é combinar as previsões de vários estimadores de base construídos com um dado algoritmo de aprendizado, a fim de melhorar a generalização /robustez de um único estimador.

Duas famílias de ensembles são geralmente distinguidas:

averaging methods: O princípio de condução é construir vários estimadores independentemente e, em seguida, calcular a média de suas previsões. Em média, o estimador combinado é geralmente melhor que qualquer um dos estimadores de base única porque sua variância é reduzida.

Exemplo: Bagging methods, Forests of randomized trees, ...

boosting methods: estimadores de base são construídos sequencialmente e um tenta reduzir o viés do estimador combinado. A motivação é combinar vários modelos fracos para produzir um conjunto poderoso.

Exemplo: AdaBoost, Gradient Tree Boosting, ...

Resultado do Ensembles

Ensemble 1 RandomForestClassifier + XGBClassifier	ROC_AUC Treinamento	Acurácia Treinamento
Fold 1 Fold 2 Fold 3	0.77176	0.70412

Ensemble 2 : RandomForestClassifier + CatBoostClassifier¶

Ensemble 2 RandomForestClassifier + CatBoostClassifier	ROC_AUC Treinamento	Acurácia Treinamento
Fold 1 Fold 2 Fold 3	0.64518	0.63608

Ensemble 3 : CatboostClassifier + XGBClassifier¶

Ensemble 3 CatBoostClassifier + XGBClassifier	ROC_AUC Treinamento	Acurácia Treinamento
Fold 1 Fold 2 Fold 3	**0.76599**	**0.69212**

Ensemble 4 LightGBMClassifier + XGBClassifier	ROC_AUC Treinamento	Acurácia Treinamento
Fold 1 Fold 2 Fold 3	**0.76370**	**0.72406**

Ensemble 5 RandomForestClassifier + XGBClassifier	ROC_AUC Treinamento	Acurácia Treinamento
Fold 1 Fold 2 Fold 3	**0.69699**	**0.63260**

Ensemble 6: CatBoostClassifier + LightGBMClassifier¶

Ensemble 6 CatBoostClassifier + LightGBMClassifier	ROC_AUC Treinamento	Acurácia Treinamento
Fold 1 Fold 2 Fold 3	**0.67987**	**0.67987**

Ensemble 7: RandomForestClassifier + XGBClassifier + CatBoostClassifier + LightGBMClassifier¶

Ensemble 7 LightGBMClassifier + CatBoostClassifier + RandomForestClassifier + XGBClassifierClassifier	ROC_AUC Treinamento	Acurácia Treinamento
Fold 1 Fold 2 Fold 3	**0.75569**	**0.70799**

CONCLUSÃO ENSEMBLE:

Este é o melhor cenário para evitar overfitting, pois potencializa as melhores qualidades de cada algoritmo. Mais uma vez o XGBClassifier e LightGBMClassifier tem uma performance melhor como podemos observar pelo que é a combinação dos 2 algoritmos e o Ensemble 7 que tem a participação dos 4 algoritmos.

Para submeter ao Kaggle, optei pelo Ensemble 7 com os 4 algoritmos.¶

Calibração/tuning do modelo final usando o GridSearchCV

Ensemble7 com GridSearchVC : RandomForestClassifier + XGBClassifier + CatBoostClassifier + LightGBMClassifier¶

Ensemble 7 com GridSearch LightGBMClassifier + CatBoostClassifier + RandomForestClassifier + XGBClassifierClassifier	ROC_AUC Treinamento	Acurácia Treinamento
Fold 1 Fold 2 Fold 3	**0.79197**	**0.70422**

Conclusão Ensemble7 com GridSearchVC

O GridSearchCV faz uma pesquisa exaustiva sobre os valores especificados para um estimador para realizar predições. Permite uma flexibidade maior, pois podemos alterar os parâmetros dos algoritmos para fazer melhores predições e verificar novos resultados.

O resultado do ROC_AUC e Acurácia obtidos com o GridSearchCV foi ligeiramente melhor do que sem GridSearchCV, o que mostra uma melhoria da predição e da prevenção do overfiting.

O ROC AUC individual para cada algoritmo melhorou significantemente conforme abaixo:

- ROC AUC LightGBM calibrado: 0.9516325614215151
- ROC AUC CatBoost calibrado: 1.0
- ROC AUC RandomForestClassifier calibrado: 1.0
- ROC AUC XGBClassifier calibrado: 1.0

CONCLUSÃO – Escolha do modelo a ser submetido

Ensemble 7 **com GridSearchVC**: RandomForestClassifier + XGBClassifier + CatBoostClassifier + LightGBMClassifier¶

```
sub = pd.DataFrame()
sub['id'] = id_test
sub['target'] = y_pred
sub.to_csv('stacked_main.csv', index=False)
pd.read_csv("stacked_main.csv")
```

O que pode ser melhorado em relação ao relatório atual¶

O que pode ser melhorado é a geração automática de relatórios criando no Markdown do próprio jupyter notebook com valores obtidos em cada algoritmo utilizado, o que ganharia tempo e os relatórios ficariam melhor apresentados como um exemplo abaixo:¶

Resultados Tabulados¶

Tabular os dados em tabelas como as abaixo automaticamente

Classificador A - {{name_classificador}}

| ROC AUC Treinamento | ROC AUC Teste | Acurácia Treinamento | Acurácia Teste |

:----------: | :----------: | :----------: | :---------- ---------: |

| {{ROC_AUC_A_Train}} | {{ROC_AUC_A_Train}} |{{ ACU_A_Train}} | {{ACU_A_Teste}} |

| {{ROC_AUC_B_Train}} | {{ROC_AUC_B_Train}} |{{ ACU_B_Train}} | {{ACU_B_Teste}} |

| {{ROC_AUC_C_Train}} |{ {ROC_AUC_C_Train |{{ ACU_C_Train}} | {{ACU_C_Teste}} |

Classificador B - {{name_classificador}}

| ROC AUC Treinamento | ROC AUC Teste | Acurácia Treinamento | Acurácia Teste |

:----------: | :----------: | :----------: | :---------- ---------: |

| {{ROC_AUC_A_Train}} | {{ROC_AUC_A_Train}} |{{ ACU_A_Train}} | {{ACU_A_Teste}} |

| {{ROC_AUC_B_Train}} | {{ROC_AUC_B_Train}} |{{ ACU_B_Train}} | {{ACU_B_Teste}} |

| {{ROC_AUC_C_Train}} |{ {ROC_AUC_C_Train |{{ ACU_C_Train}} | {{ACU_C_Teste}} |

Classificador C - {{name_classificador}}

| ROC AUC Treinamento | ROC AUC Teste | Acurácia Treinamento | Acurácia Teste |

:----------: | :----------: | :----------: | :---------- ---------: |

| {{ROC_AUC_A_Train}} | {{ROC_AUC_A_Train}} |{{ ACU_A_Train}} | {{ACU_A_Teste}} |

| {{ROC_AUC_B_Train}} | {{ROC_AUC_B_Train}} |{{ ACU_B_Train}} | {{ACU_B_Teste}} |

| {{ROC_AUC_C_Train}} |{ {ROC_AUC_C_Train |{{ ACU_C_Train}} | {{ACU_C_Teste}} |