

Machine Learning Engineer Nanodegree

Capstone Project

Marcelo Hanones
setembro, 2018.

I. Definition

Project Overview

O aprendizado de machine learning compreende o domínio de diferentes áreas. São diversos conceitos que se ordenam e se completam para justificar hipóteses sobre os dados. Este projeto foca na organização necessária para o acúmulo de informação durante todos os passos da análise. Houve bastante foco no caminho percorrido, e não somente na solução final.

Problem Statement

O desafio é fazer o computador capturar determinadas características de um texto que estejam relacionadas a uma classificação previa dada a este texto. O que exatamente faz com que determinado review tenha sido classificado como positivo ou negativo ? A resposta dessa pergunta é parte de uma aplicação de NLP chamada análise de sentimento.

As tarefas macro envolvidas são:

- Baixar o dataset
- Pre-processar e vetorizar o dataset
- Treinar o modelo
- Otimizar o modelo

Metrics

A métrica usada é o `roc_auc`. Essa métrica é na verdade o conjunto dos resultados obtidos ao variar o valor de `threshold` $[0,1]$. Os resultados são mostrados num gráfico de curva que mostra o efeito que a variação de `threshold` ocasiona na proporção dos *true positive* com os *false positive*. A escolha dessa métrica traz consigo a necessidade do algoritmo ser aderente ao método `predict_proba`.

II. Analysis

Data Exploration

O dataset contém 25.000 entradas divididas em :

15.000 treino

5.000 validação

5.000 teste

O arquivo é chamado labeledTrainData.csv e possui header e três colunas:

id - identificador único

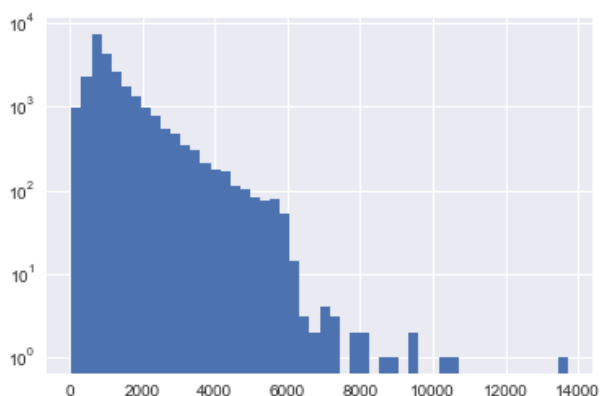
sentiment - contém valores binários (0 and 1) para positivo e negativo

review - contém o texto escrito pelo usuário.

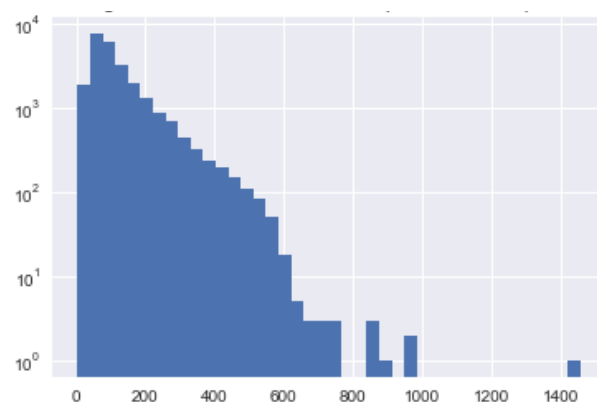
Exploratory Visualization

Os gráficos abaixo mostram como a limpeza é parte importante do processo. Entre a versão *raw* , que contém pontuação e *stop_words*, e a *versão final* existe um volume 10 vezes menor de palavras.

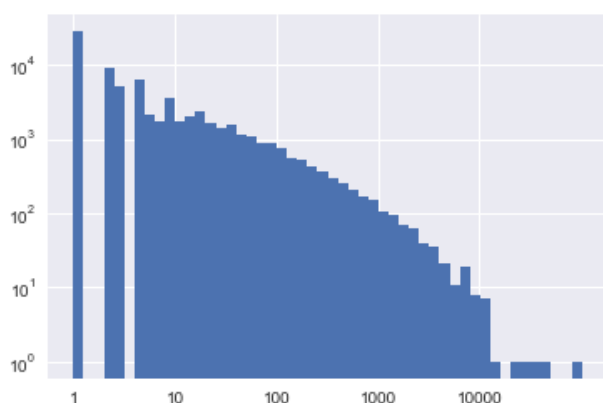
Histogram of Word Count
with stop words and punctuation



Histogram of Word Count
without punctuation nor stop words



Histogram of Word Frequencies
without stop words



O histograma da contagem de vezes que cada palavra se repete no texto mostra que muitas palavras ocorrem poucas vezes.

Abaixo seguem as 20 palavras mais comuns para os reviews positivos e para os negativos. Fica claro que as palavras mais comuns em ambos reviews são basicamente as mesmas. Isso mostra que o processo de determinar quais palavras fazem mesmo diferença é vital para aumentar o poder de predição do modelo. Em outras palavras, a maioria das features são dispensáveis.

Positive			Negative		
	word	freq		word	freq
1	movie	24969		film	20940
2	film	19219		movie	19078
3	one	13138		one	13657
4	like	11241		like	9040
5	even	7691		good	7724
6	good	7423		story	6779
7	bad	7401		time	6516
8	would	7036		great	6419
9	really	6263		well	6411
10	time	6211		see	6027
11	see	5452		also	5551
12	story	5209		really	5475
13	much	5077		would	5400
14	get	5040		even	4964
15	people	4807		first	4757
16	make	4722		much	4687
17	could	4685		people	4480
18	made	4541		best	4320
19	first	4307		love	4302

Algorithms and Techniques

O processo exploratório para achar o melhor modelo fez uso de algoritmos com diferentes approaches. Dessa forma temos algoritmos paramétricos e não-paramétricos e podemos ver como cada tipo reage aos dataset tanto no quesito desempenho, tendência a overfitting e tempo de processamento.

Os algoritmos escolhidos são:

- Decision Trees
- Random Forest
- Adaptative Boosting
- ExtraTrees
- LightGBM
- KNN
- LinearSVC
- Logistic Regression
- Naive Bayes
- Stockastic Gradient Descent

Decision Trees - faz a representação de uma tabela de decisão com estimativas e probabilidades associadas aos resultados de cursos de ação que competem entre si.

Random Forest - é um método de ensemble que combina os resultados de diversas Decision Trees. Pode representar outras relações dos dados além das lineares.

Adaptative Boosting - é um meta-algoritmo, é utilizado para aumentar a performance de outros algoritmos de aprendizagem. O AdaBoost é adaptável no sentido de que as classificações subsequentes feitas são ajustadas a favor das instâncias classificadas negativamente por classificações anteriores.

ExtraTrees - é uma variação do Random forest onde a cada passo todo o sample é utilizado e as *decision boundaries* são escolhidas de forma randômica, daí o nome "Extremely randomized trees" .

LightGBM - é um algoritmo de Boosting baseado em árvore que tem como característica crescer as árvores verticalmente ao invés de horizontalmente. O termo 'Light' vem da sua alta velocidade eficiência em grandes bases de dados.

KNN - faz a classificação de um ponto de acordo com a classificação existente nos pontos vizinhos.

LinearSVC - este algoritmo é bastante popular no meio devido aos bons resultados. Um ponto negativo é que ele demanda bastante memória e possui diversos hyper-params para otimizar.

Logistic Regression - é um classificador binário usado para dividir os dados quando estes são linearmente separáveis. São algoritmos mais rápidos e representam a solução mais simples.

Naive Bayes - esse algoritmo é bastante simples e possui poucos hyper-params. É comumente conhecido em aplicações de classificação como filtro de spam. É baseado na premissa de que as features são independentes uma da outra.

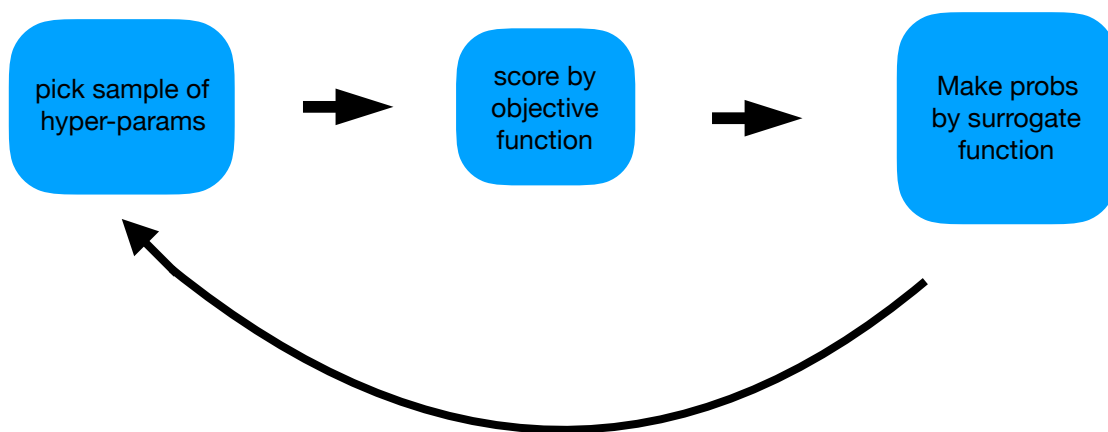
Stockastic Gradient Descent - In Gradient Descent optimization, we compute the cost gradient based on the complete training set, and it is costly for large datasets. The term "stochastic" comes from the fact that the gradient based on a single training sample is a "stochastic approximation" of the "true" cost gradient.

Todos estes algoritmos serão submetidos à otimização de parâmetros. As técnicas usadas para otimização dos parâmetros são:

- cross validation
- Validation set
- Métodos bayesianos via biblioteca HyperOpt
- Métodos “exaustivos” via biblioteca PARFIT



O métodos bayesianos usam probabilidades baseadas no passado para determinar as melhores escolhas futuras. Esses métodos prometem rapidez uma vez que calcular as probabilidades é menos custoso do que processar todo o dataset a cada iteração. Esses métodos operam de forma oposta ao métodos exaustivos, que processam todas as combinações do grid sem nenhuma tipo de inteligência organizando o processo de escolha. Métodos exaustivos se assemelham aos algoritmos conhecidos como brute force.

Em termos mais práticos, a biblioteca HyperOpt escolhe no grid uma combinação qualquer de *hyper-params* e roda essa combinação como parâmetro de entrada de uma função chamada *objective*. Essa função *objective* produz um *score*. Uma outra função conhecida como ‘*surrogate function*’ usa esse *score* como informação para, depois de um número ótimo de iterações, produzir um modelo probabilístico do *score* previsto de cada *sample*.



Benchmark

O modelo de referência para este projeto é o patamar de 95% de *auc_score*. Este *score* representa aproximadamente os 170 melhores resultados da competição.

174	▼ 34	Caesar11		0.95049	2	3y
175	new	newprolab.com.vladimir.litvin...		0.94949	11	3y

Este *score* deve vir acompanhado de uma implementação com tempo de processamento razoável e uma *learning_curve* sem overfitting.

III. Methodology

Data Preprocessing

Esse foi o pipeline basico aplicado a todos os modelos.

```
pipe = Pipeline([
    ('col', Col_Extractor(['review'])),
    ('prep', Preprocessor()),
    ('to_array', To_array()),
    ('vec', _____()),
    ('mdl', _____())
])
```

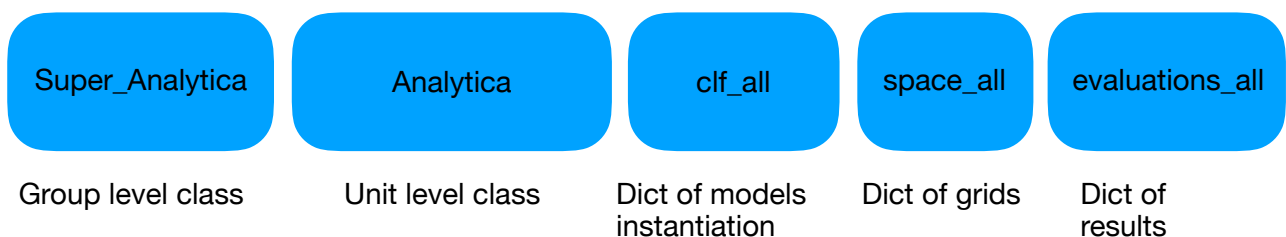
Analizando os três primeiros elementos do pipeline:

1 - Embora o dataset tenha somente uma única coluna, optei por uma usar um *column extractor* para simular arquiteturas mais complexas onde uma(s) transformação é aplicada em um grupo de colunas específicos.

2 - O objeto Preprocessor realiza a limpeza dos dados, retirando html e pontuação mas mantendo emoticons.

3 - O objeto to_array transforma o dataframe em um array para ser processado pelos métodos subsequentes. Desta forma o dataset permanece no formato DataFrame durante toda a fase de limpeza, tomando vantagem do ferramental que DataFrames dispõem para processamento de strings.

Implementation



Toda a implementação foi feita com base nos cinco elementos acima, duas classes e três dicionários.

Os dicionários 'clf_all' e 'space_all' contem parâmetros escolhidos pelo usuário. Respectivamente os classificadores escolhidos e os grids para otimização dos *hyper-parameters* desses classificadores. O dicionário 'evaluations_all' seria o equivalente a um banco de dados central que armazena as informações produzidas pelo processamento do código. Todas as funções acessam esse dicionário seja para ler as informações ou gravar novas informações. Por exemplo quando um modelo é carregado via pickle, este é inserido neste dicionário ficando a disposição de todas as funções da classe.

Quanto as classes, a ideia foi ter uma classe que manipule grupos de objetos para rotinas mais generalistas e outra classe para trabalhar mais a fundo um único objeto. Por exemplo, a classe para plotar o ranking score de um grupo de modelos seria uma classe generalista. E a classe mais específica seria para otimização de parâmetros onde seria necessário relacionar pipelines e grids por exemplo.

Classe Super_Analytica

A classe `Super_Analytica` foi pensada para facilitar o manuseio de grupos de modelos. Diferente da classe `Analytica`, esta classe não permite grandes customizações. Basicamente esta classe aplica um mesmo processo a um grupo de modelos. Qualquer função que rode sempre com os mesmos parâmetros pode fazer parte desta classe. Por exemplo, no primeiro notebook é feita uma *cross-validation* com todos os algoritmos. Pela *cross-validation* ser uma função sem customizações, que roda sempre em estado “default”, a implementação foi realizada na classe `Super_Analytica`. Numa única linha de código, a mesma função `Evaluate_cv` é aplicada a todos os algoritmos, dando origem a um número igual de modelos. Essa função ainda registra todos os scores num dicionário central com a opção de serializar os resultados em disco (pickle).

Um exemplo de uso seriam os comandos para plotar a learning curve de um grupo de modelos:

```
master = Super_Analytica()

master.plot_learning_curve(chosen= GrupoDeObjetos )
```

Classe Analytica

A classe `Analytica` foi pensada para facilitar o parameter tuning, portanto opera em nível individual possibilitando a customização e cruzamento de diversos *hyper-parameters*. Cada *instance* dessa classe funciona como um *container* de todas as informações necessárias para modularizar o processo de otimização.

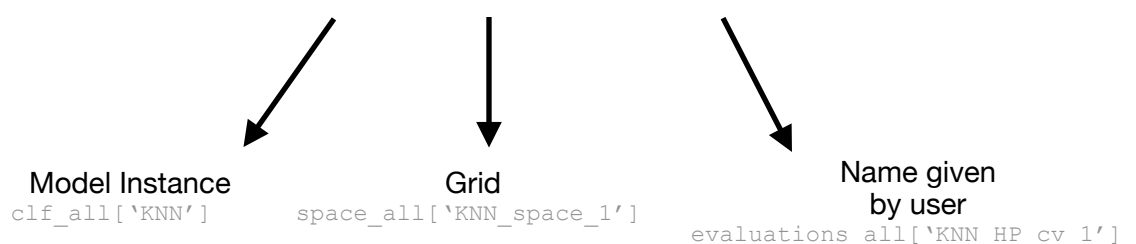
A declaração dessa classe exige:

`instance` - model instance like `SVC()`. Deve corresponder a uma ‘key’ do dicionário ‘`clf_all`’.

`space_all` - é um dicionário com dicionários que contêm os parâmetros a serem otimizados. Deve corresponder a uma ‘key’ do dicionário ‘`space_all`’.

`my_name` - todos os scores produzidos pela função são armazenados num dicionário que é armazenado dentro de uma *Class Variable* que funciona como um dicionário central chamado `evaluations_all`. `my_name` corresponde a uma key de `evaluations_all`.

```
md_KNN = Analytica('KNN', 'KNN_space_1', 'KNN_HP_cv_1')
```



No total são cinco implementações diferentes de funções com objetivo de medir os scores dos modelos. Dessas cinco, a mais simples é a função `Evaluate_CV` localizada na classe `Super_Analytica`. O restante são funções para otimização de *hyper-params*, localizados na classe `Analytica` e `Analytica_2` e seguem melhor descritas abaixo:

`evaluate_HP_cv` - implementação da biblioteca `hyperOpt` com cross-validation na função *objective*. Processa todo o dataset on-the-fly, fold a fold.

`evaluate_HP_val` - implementação da biblioteca `hyperOpt` com set de validação na função *objective*. Esta função foi criada na classe chamada `Analytica_2` que pre-processa o dataset de antemão. Essa classe é melhor descrita mais abaixo.

`evaluate_HP_IGBM` - implementação da biblioteca `HyperOpt` usando `LihgtGBM` na função *objective*.

`evaluate_Parfit` - implementação da biblioteca `PARFIT`. Essa biblioteca é uma caixa-preta e sua implementação é bastante standard.

De forma geral as funções de otimização realizam as seguintes rotinas:

Hyper-params Optimisation
 Fit best_params on whole training set
 Predict and Score on test set
 Send scores and other information to 'evaluations_all'
 Pickle

Dessa maneira diferentes pipelines podem ser testados com diferentes grids de forma organizada.

Name	Class	Tune classif. params	Tune all pipe params	função
evaluate_CV	Super_Analytica	0	0	<code>evaluate_CV(self, pipe, chosen, suffix='', pkl_W=False, vectorize_first=False):</code>
HP_cv	Analytica	1	1	<code>evaluate_HP_cv(self, pipe, pkl_W=False)</code>
HP_val	Analytica_2	1	0	<code>Evaluate_HP_val(self, classifier, space, my_name, pkl_W=False):</code>
HP_PARFIT	Analytica	1	0	<code>evaluate_Parfit(self, pipe, pkl_W=False):</code>
HP_LightGBM	Analytica	1	0	<code>evaluate_HP_IGBM(self, pipe, pkl_W=False):</code>

Enquanto na função *HP_cv* todos os parâmetros de todos os métodos do pipeline podem ser otimizados, na função *HP_val* somente os parâmetros do classificador podem ser otimizados.

Class Analytica_2

Realizar o preprocessing do dataset on-the-fly permite liberdade de otimizar tanto os parâmetros do modelo como também os parâmetros dos métodos de preprocessing. Em contrapartida consome maiores recursos computacionais.

No intuito de evoluir no processo de otimização, criei uma variação da classe Super Analytica chamada Analytica_2.

Esta Nova Classe foi criada para facilitar a experimentação de formas diferentes de pre-processamento e redução de dimensionalidade. Ao invés de processar o dataset on-the-fly via pipeline, esta classe realiza a vetorização de antemão no momento em que a *class instance* é criada. A partir disso o dataset passa a ser uma variável interna compartilhada por todos os métodos da classe. A rotina de vetorização é realizada uma única vez, logo os tempos de processamento reduziram bastante

A declaração dessa classe ocorre de forma diferente das anteriores.

Cada instância é associada a um único pipeline como atributo. O método `make_class` faz uso desse pipeline, vetorizando e serializando os dados via pickle para uso posterior por qualquer método da classe sem a necessidade de refazer todo o processo novamente. O método `read_data` faz a leitura do dataset já armazenado em disco.

```
md_10 = Analytica(pipe_10)
md_10.make_data()
md_10.read_data()
```

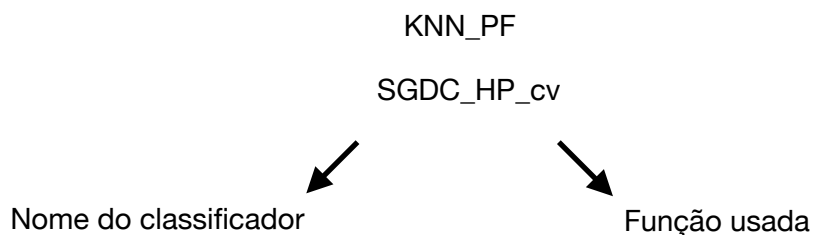
```
md_10.evaluate_HP_val
```

Pipelines

Uma observação importante sobre os pipelines. O pipeline é um elemento bastante útil em arquiteturas onde há a necessidade de pre-processing como ocorre em NLP. No intuito de facilitar o processo de experimentação eu optei por declarar os pipelines sem o último elemento, sem o classificador. Dessa maneira o pipeline 'incompleto' funciona como um *core container* de todos os métodos de pre-processing e vetorização reunidos num único objeto que pode ser aproveitado como parâmetro inicial das funções de otimização. No momento que qualquer uma das funções de otimização são executadas o pipeline sofre o *append de um classificador*, passando então a estar completo e pronto para os métodos `.fit` e `.predict`.

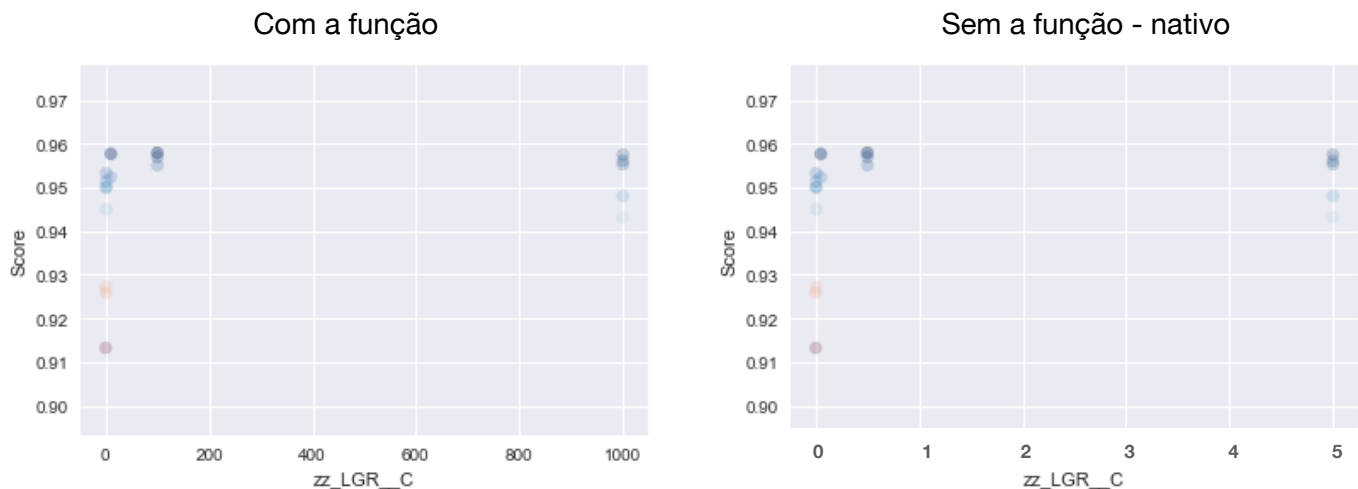
Naming

Todos os modelos são armazenados em `evaluations_all` após serem processados pelas funções de otimização de parâmetros listadas acima. Esses modelos recebem um nome único descrito abaixo. O nome é uma combinação do classificador com a função.



helper_translate e helper_translate_2

Essas duas funções ajudam na visualização da evolução dos scores no processo de otimização dos *hyper-params*. A biblioteca oferece uma funcionalidade chamada *choice* que permite que uma lista de valores arbitrários 'L' seja usada como lista de parâmetros a serem otimizados. De forma nativa a biblioteca retorna como resultado a posição(index) da lista que obteve o melhor score e não o valor em si. Como pode ser visto no eixo x dos gráficos abaixo, na esquerda vemos os valores reais, na direita valores enumerados de 0, len(L)



```
param['LGR_HP_1'] = {  
    .  
    .  
    .  
    .  
    'LGR_C': hp.choice('zz_LGR_C', [1.e-1, 1.e+0, 1.e+1, 1.e+2, 1.e+3])}
```

Para que essa função seja utilizada basta declarar o parâmetro no pipeline usando o sufixo 'zz_' conforme exemplo acima. Uma vez que a implementação HP_cv permite que todos os parâmetros do pipeline sejam otimizados, essas funções facilitam a leitura dos resultados quando a quantidade de parâmetros sendo otimizados simultaneamente aumenta.

Refinement

De forma geral, o processo de refinamento foi dividido em 6 notebooks organizados da seguinte maneira:

Master 1 - Cross-validation - Class Super_Analytica

Master 2 - Parameter tuning with HyperOpt on cross validation - Class Analytica

Master 3 - Optimisation with Parfit - Class Analytica

Master 4 - LightGBM - Class Analytica

Master 5 - Parameter tuning with HyperOpt on validation set - Class Analytica_2

Master 6 - Analise Geral

No notebook 1 foi feito um teste entre dois métodos de redução de dimensionalidade utilizando Logistic Regression, que é um algoritmo que realiza todo o processamento com rapidez. Os métodos escolhidos foram TruncatedSVD e SelectKBest, respectivamente um método complexo e um simples.

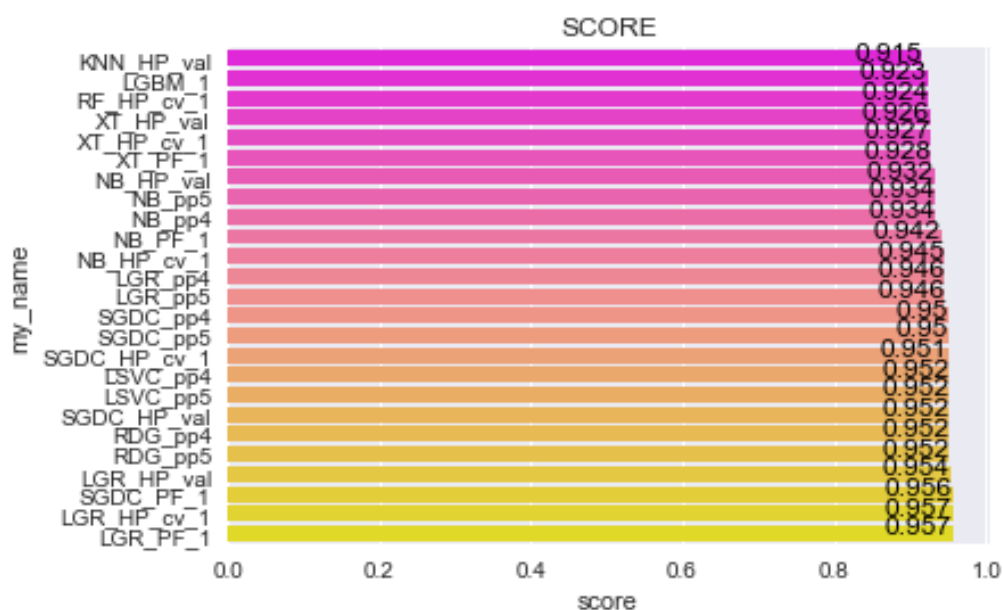
O método SelectKBest tomou um tempo 7 vezes menor e foi o método de redução de dimensionalidade escolhido nas análises seguintes para os algoritmos não paramétricos.

Para os algoritmos paramétricos nenhum método de redução de dimensionalidade foi utilizado nessas primeiras análises. A premissa de que esses algoritmos 'ignoram' a quantidade de dados foi a justificativa para não usar nenhum método de redução de dimensionalidade. Embora a execução desses algoritmos seja rápida, houve overfitting deixando claro que a redução de dimensionalidade poderia ser útil para tornar o dataset mais conciso e assim reduzir o overfitting. Abaixo segue o pipeline do modelo inicial e do modelo final de Logistic Regression:

```
pipe_4 = Pipeline([
    ('col', Col_Extractor(['review'])),
    ('prep', Preprocessor()),
    ('to_array', To_array()),
    ('tf-idf_vec', TfidfVectorizer())
])

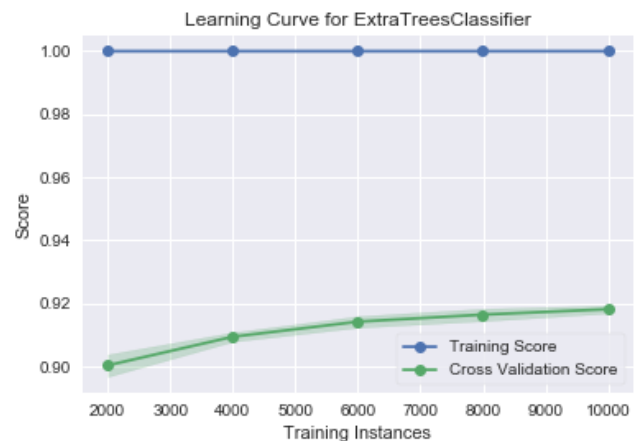
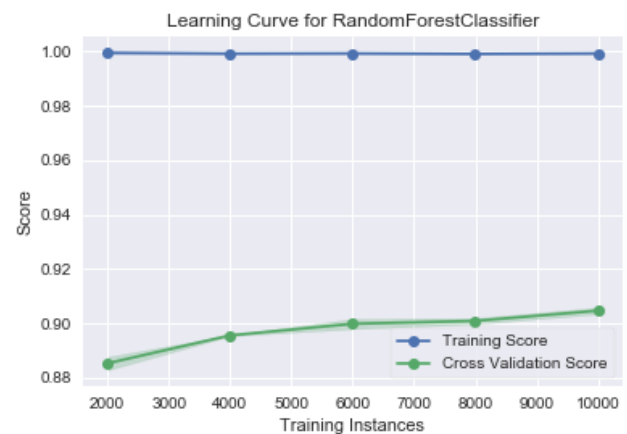
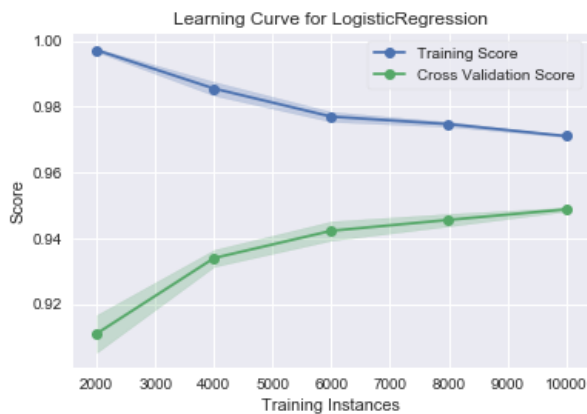
pipe_10 = Pipeline([
    ('col', Col_Extractor(['review'])),
    ('prep', Preprocessor()),
    ('to_array', To_array()),
    ('vec', TfidfVectorizer(stop_words='english')),
    ('svd', TruncatedSVD(n_components=800))
])
```

De forma geral os algoritmos não paramétricos não obtiveram bons desempenhos nesse dataset.

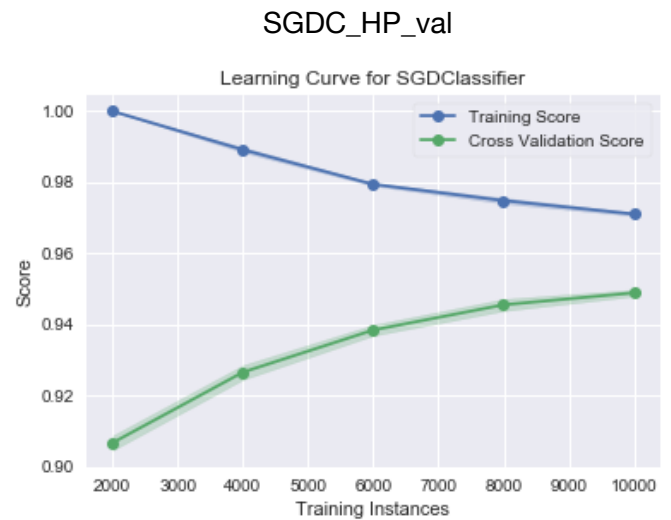
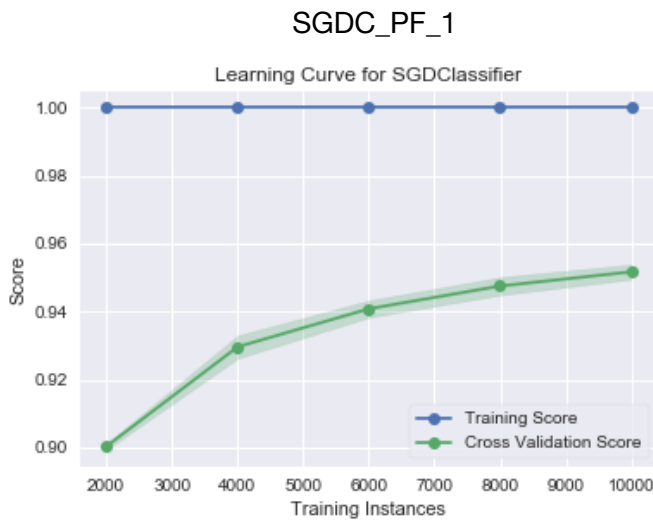


Além de baixo score, todos mostraram tendencia ao overfitting. Comparando as learning_curve de ambos grupos fica visível a diferença.

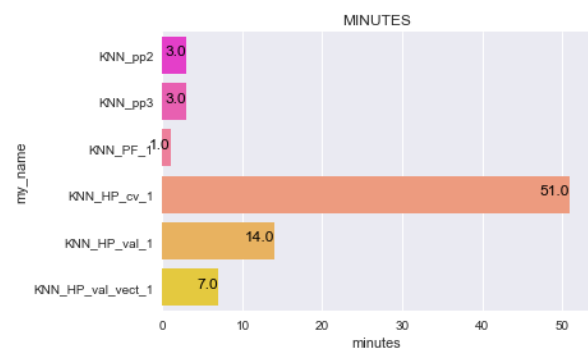
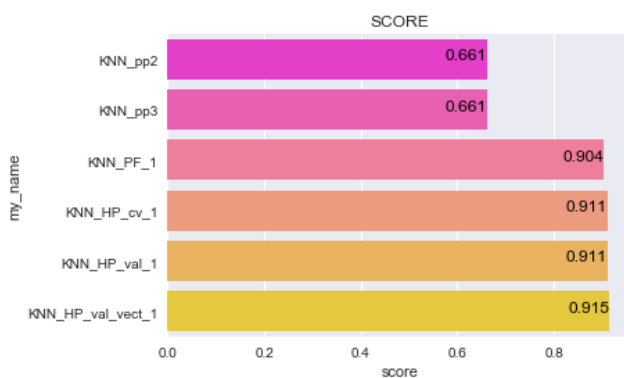
Paramétricos → LinearSVC, LGR, NB, SGDC. Não paramétricos → DT, ADA, RF, XT, KNN



A learning_curve do algoritmo SGDC mostra a evolução ocorrida entre cada implementação. A redução no gap entre ambos scores mostra que aquilo que o modelo aprendeu durante a fase de treinamento teve quase o mesmo desempenho quando aplicado a ambos datasets. Ou seja, o modelo tende a ter desempenhos iguais em ambos datasets. Em outras palavras o modelo tende a errar na mesma quantidade em ambos datasets. Isso sugere que os datasets possuem características semelhantes e sugere também que durante a fase de treino o algoritmo aprendeu tudo que ele poderia aprender. Digamos que o algoritmo ‘deu tudo o que podia’ para este dataset, ou esta perto disso.



Alguns modelos foram mais sensíveis à otimização de parâmetros do que outros. De todos os modelos o KNN foi o que mais aumentou o score com otimização de parâmetros, de 0.6 para 0.9.

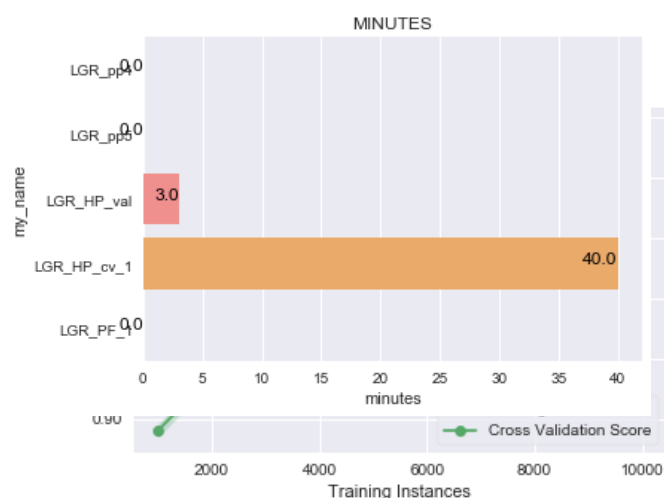
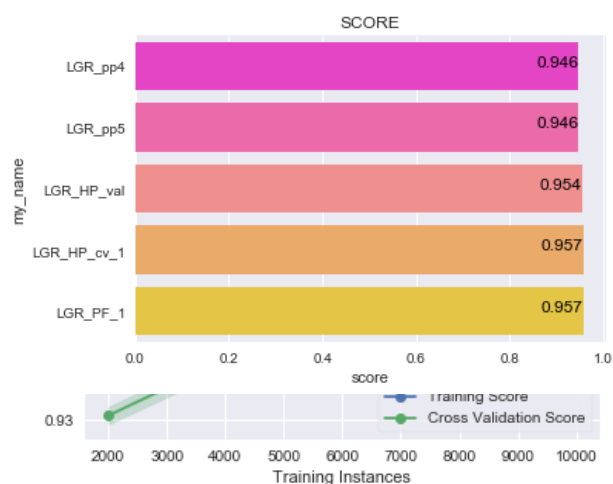


IV. Results

Model Evaluation and Validation

Toda as análises dos modelos comparam o score com as learning_curve. Diferentes implementações obtiveram scores, tempos e learning_curves diferentes.

O algoritmo Logistic Regression obteve o melhor score em sua versão default, porem a learning_curve mostra overfitting, logo a capacidade de generalização é deficiente. Já a versão otimizada obteve um score menor porem mais “real” com maior capacidade de generalização.



Justification

O objetivo maior de um algoritmo de machine learning é promover a generalização para dados nunca vistos. O modelo LGR_HP_val foi escolhido como o melhor pois obteve o score de 0.954 com uma learning curve mostrando um estágio avançado de convergência entre os scores de treino e teste.

A implementação da classe Analytica_2 reduziu o tempo de processamento realizando toda a preparação do dataset de antemão e não durante o processo de fit / predict.

V. Conclusion

Reflection

A escolha do melhor parâmetro não livra o modelo do overfitting. Pelo contrario, um determinado parâmetro pode ocasionar aumento de test score mas este deve ser analisado em conjunto com os scores de treino e validação. No fundo, o mais importante é a relação entre os três, quanto menor a diferença melhor.

Usando o modelo escolhido podemos fazer algumas simulações de textos livremente produzidos. Seguem algumas simulações e a classificação dada pelo modelo. (Notebook 6)

Simulated review	Classification
the movie was great. it is awesome	Pos
movie is boring. I almost slept	Neg
movie is boring at the beggining, but the effects are great and overall feeling is great too.	Pos
movie is boring at the beggining and the effects are poor. But the story os compelling and overall feeling is great.	Neg

As duas primeiras frases tem entendimento direto devendo receber a classificação positiva e negativa, respectivamente. Já a terceira e quarta frases tem um entendimento menos claro, mas mesmo assim o modelo fez uma classificação condizente.

Improvement

Um aspecto da engenharia da ferramenta que pode ser melhorado é a importação de informações localizadas em outros arquivos.

A medida que as análises foram se desdobrando em sequencias de notebooks, ficou tedioso e confusa a tarefa de copiar as variáveis globais e as classes para cada novo notebook. Seria mais fácil declarar essas informações num arquivo separado e importar esse arquivo a cada notebook.

A solução foi declarar as variáveis globais num notebook e as classes em outro notebook e utilizar o método %run para “importar” essas informações. Seria mais elegante realizar a importação das classes via o método import. Porém, o método %run foi a única maneira que consegui para que as variáveis globais fossem carregadas no mesmo escopo das classes. Eu tentei utilizar o método import (from ____import ____) mas as classes não reconheceram as variáveis globais. Ainda estou pesquisando para entender mais esse problema.