

Universidade Federal de Pernambuco

Centro de Informática - CIn

Disciplina: IF699 - APRENDIZAGEM DE MÁQUINA

Docente: Francisco de Assis Tenorio de Carvalho

Equipe:

Amadeo Tato Cota Neto

Arthur Andrade Lacet Florencio

Bruno Henrique dos Santos Marques

Caue Addae da Silva Gomes

Matheus Paixao Gumercindo dos Santos

**PROJETO APRENDIZAGEM DE MÁQUINA 2022.2**

## 

## Apresentação do dataset

* O conjunto de dados foi extraído de 800 imagens da "Bíblia de Ávila". Foi constatado que houve a presença de 12 copistas na escrita da bíblia em que as páginas escritas por cada copista não são igualmente numerosas. Cada padrão contém 10 características.
* O modelo que deve ser criado consiste em associar cada padrão de escrita a um dos 12 copistas (rotulados como: A, B, C, D, E, F, G, H, I, W, X, Y).

## Analisando as variáveis

A nossa variável alvo (target vector) é a **Class** que corresponde a associação de cada padrão para um dos 12 copistas, representados pelas letras: A, B, C, D, E, F, G, H, I, W, X, Y.

O conjunto de recursos projetados para distinção do escriba pode ser dividido em três grupos, principalmente no que diz respeito ao layout da página ou à interação entre a caligrafia e o espaço disponível na página.

### Primeiro grupo - propriedades geométricas de toda a página

* O atributo **Intercolumnar Distance** indica a distância intercolunar que é utilizada por um copista na hora de sua escrita, é uma avaliação feita de forma vertical
* O atributo **Upper Margin** indica a margem superior dada por um copista
* O atributo **Lower Margin** indica a margem inferior dada por um copista

As features desse grupo não são muito distintas para um copista individual, mas pode ser muito útil para destacar diferenças cronológicas e/ou tipológicas.

### Segundo grupo - Exploração de cada coluna na área escrita

* O atributo **Exploitation** é medido através de um coeficiente que calcula o quanto a coluna pega como amostra está preenchida de tinta
* O atributo **Row Number** indica o número de linhas que contém na coluna que foi pega como amostra na área de escrita

Nesse segundo grupo, tais características podem ser consideradas como uma medida da habilidade do escritor e pode ser muito útil para distinção do escriba.

### Terceiro grupo - Forma como o escriba distribuía o texto em cada linha

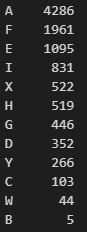
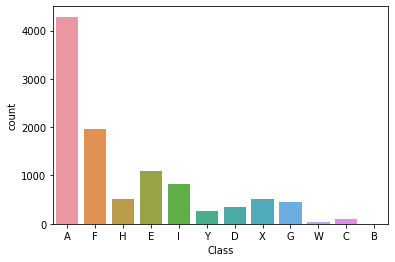
* O atributo **Weight** informa o quanto a linha está preenchida com tinta (medida de espessura do traço)
* O atributo **Peak** number é uma estimativa razoável do número de caracteres em uma linha
* O atributo **Modular Ratio** é uma taxa que calcula o tamanho horizontal pelo tamanho vertical de um caractere (largura/altura)
* O atributo **Interlinear Spacing** informa o espaço que é dado entre as linhas
* O atributo **Modular Ratio/Interlinear Spacing Ratio** é uma taxa que é obtida dividindo-se esses 2 parâmetros previamente especificados

As features desse grupo caracterizam a maneira de escrita adotada por um único escriba em um contexto específico, mas podem também sugerir distinções geográficas e/ou cronológicas.

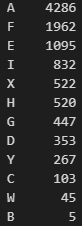
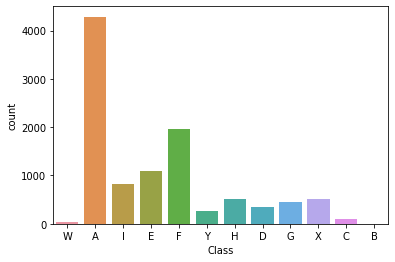
## Analisando o dataset

Obtendo informações mais detalhadas sobre as colunas (tipo e contagem) para o dataset de treinamento, pode-se observar que o nosso dataset possui 10430 linhas e 11 colunas. Já o dataset de teste tem os atributos idênticos ao dataset de treino (como esperado) e possui 10437 linhas (exemplos). Além disso, percebe-se que tanto o dataset de teste quanto o de treino não possuem valores ausentes ou NaN.

Vamos agora verificar como está o balanceamento do nosso dataset levando em consideração a variável alvo de ambos os datasets. Primeiro faremos para o dataset de treinamento.

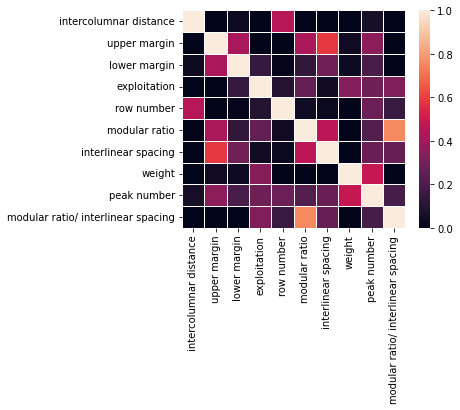
 

Agora aplicamos ao dataset de teste

Podemos observar a partir da nossa análise que o nosso dataset possui classes com muitos exemplos e algumas classes possuem pouquíssimos, podendo destacar a classe B que é a que possui menos exemplos (5 exemplos). Além disso, podemos observar que a quantidade de exemplos no dataset de treinamento e teste para cada classe são similares.

Podemos usar agora a matriz de correlação ao nosso favor para checar a correlação entre as features:



Podemos constatar que há uma correlação maior entre peak number e weight além de modular ratio e modular ratio/interlinear spacing (como já foi apontado na descrição dos atributos). Podemos também tirar como conclusão que apesar da divisão ser entre os atributos modular ratio e interlinear spacing, o atributo de modular ratio influencia mais, sendo mais discriminante. Outra correlação que podemos constatar é a de interlinear spacing e upper margin.

Para os experimentos seguintes, usamos uma seed fixa de 32642 para controlar a aleatoriedade.

## Classificação usando Árvore de Decisão

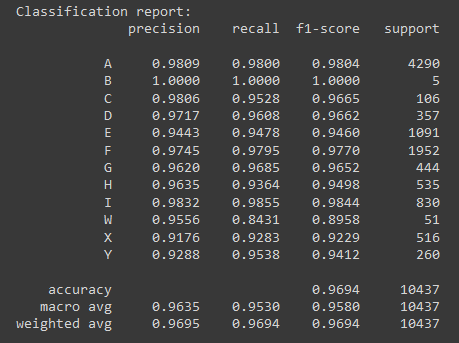
### Modelo base

**Parâmetros:**

* random\_state=seed

Treinaremos um modelo base, sem definir hiperparâmetros, e obteremos os resultados preditos do treinamento usando o dataset de teste.

### Análise dos resultados do modelo base



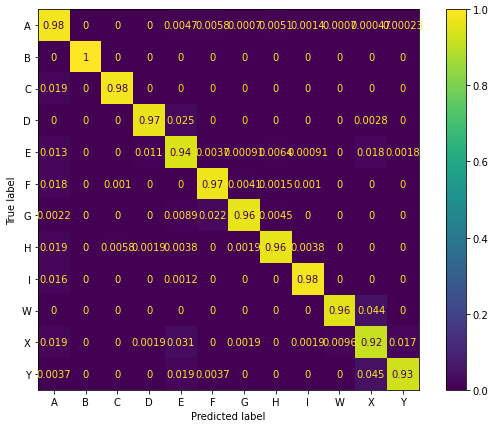
**Resultados para cada classe:**

* Percebemos uma boa média geral nos valores do Precision Score, com todas acima de 0.9, ressaltando o resultado perfeito para a classe "B".
* Para os resultados de Recall Score e F1-Score, os resultados foram semelhantes aos do Precision Score, exceto para a classe "W" que obteve resultados abaixo de 0.9 para o Recall e F1-Score, provavelmente, por uma alta quantidade de Falsos Negativos no cálculo.

**Resultados gerais:**

* Obtivemos uma acurácia satisfatória de ~0.97.
* Também obtivemos médias macro satisfatórias de Precision = ~0.96, Recall = ~0.95 e F1-Score = ~0.96.

### Matriz de confusão do modelo base



Podemos perceber que houve um desempenho relativamente pior em distinguir as classes E, W, X e Y.

### Otimização do modelo usando GridSearchCV

Parâmetros que atualizaremos no GridSearchCV para o DecisionTreeClassifier:

* criterion: função que mede a qualidade do split dos nós.
  + gini: para a Gini impurity;
  + log\_loss e entropy: ambos para o ganho de informação Shannon.
* splitter: estratégia usada para separar cada nó.
  + best: escolhe o melhor split;
  + random: escolhe o melhor split aleatório.
* max\_depth: profundidade máxima da árvore.
* ccp\_alpha: custo de complexidade que permite realizar um método de poda da árvore, o Minimal Cost-Complexity Pruning (post prunning). A sub-árvore com o maior custo de complexidade que é menor que ccp\_alpha será escolhida.
  + 0.0: sem poda.
* random\_state: controla a aleatoriedade do estimador. Usaremos a nossa Seed setada previamente.

Para escolher os melhores parâmetros, usaremos o GridSearch, que faz uma busca exaustiva no espaço de valores usando validação cruzada(Cross-Validation), eliminando a necessidade de se separar uma parte do conjunto de treino para realizar a validação do modelo. Usaremos 5 folds para o Cross-Validation do modelo, assim permitindo com que pelo menos um elemento da classe B fique em cada fold.

### Obtendo os melhores parâmetros pelo GridSearchCV

**Parâmetros:**

* dtc
  + dtc = DecisionTreeClassifier()
* params
  + params = {'criterion':['gini', 'entropy', 'log\_loss'],

'splitter':['best','random'],

'max\_depth':[None, 2, 5, 10, 15, 20],

'ccp\_alpha':[0.0, 0.0000001, 0.00001, 0.001, 0.01, 0.1, 0.5, 0.75],

'random\_state': [SEED]}

* cv=5
* refit=True

**Melhores parâmetros obtidos:**

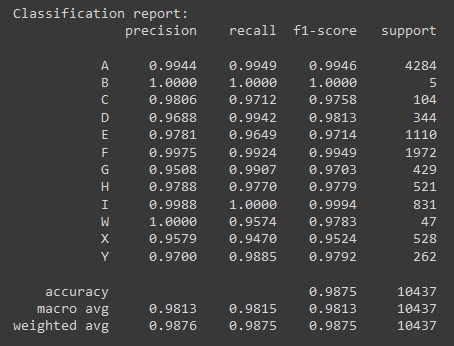
* ccp\_alpha: 0.0,
* criterion: 'entropy',
* max\_depth: None,
* splitter: 'best'

### Treinando novo modelo com os melhores parâmetros do GridSearchCV

**Parâmetros:**

* ccp\_alpha: 0.0,
* criterion: 'entropy',
* max\_depth: None,
* random\_state: 32642,
* splitter: 'best'

### Análise dos resultados do novo modelo otimizado pelo GridSearchCV



**Comparando resultados dos modelos Pré e Pós GridSearchCV.**

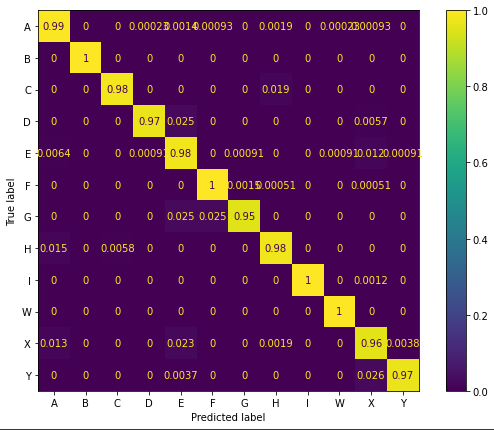
**Resultados para cada classe:**

* Houve uma melhora satisfatória nos resultados de ambos Precision, Recall e F1-Score.
* A classe I no modelo otimizado pelo grid search acabou com 100% de acerto no teste para o Recall.
* A classe W que no modelo pré-grid search tinha resultados abaixo de 0.9 para o Recall e F1-Score, agora possui em todos os scores resultados acima de 0.9. E com a ressalva de que conseguiu 100% de acerto no teste para o Precision Score.
* A classe X agora ocupa o papel da classe com os menores scores, ainda que acima de 0.9.

**Resultados gerais:**

* Obtivemos uma melhora de ~0.97 para ~0.99 na acurácia.
* Os resultados para as médias macro também obtiveram uma melhora de:
  + Precision = ~0.96 para ~0.98;
  + Recall = ~0.95 para ~0.98;
  + F1-Score = ~0.96 para ~0.98.

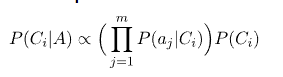
### Matriz de confusão do novo modelo otimizado pelo GridSearchCV



Podemos perceber visualmente a melhora geral na distinção das classes. É notável que o modelo passou a classificar melhor para todas as classes exceto a classe G, que teve uma queda de 0.1.

## Classificação usando Naive Bayes

Os classificadores Naive Bayes tomam como base a seguinte modificação do teorema de Bayes:



Isso significa que assumimos a independência entre as features (por isso o termo "naive"). Dessa forma, como vimos anteriormente que os atributos do dataset não são linearmente correlacionados, não será necessário realizar um tratamento especial para lidar com a correlação.

### Escolha do Classificador

Existem os seguintes classificadores Naive Bayes no scikit:

* Gaussian
* Bernoulli
* Categorical
* Complement
* Multinomial

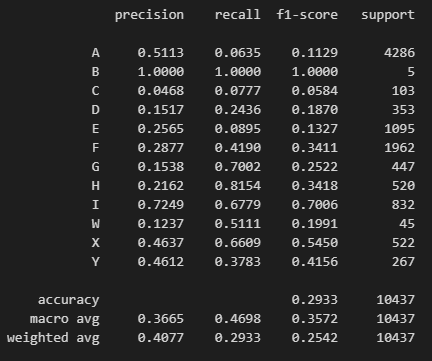
O classificador selecionado foi o Gaussian já que ele é apropriado para dados contínuos, enquanto os outros são para outros tipos de dados (e.g. discreto, binário, etc...)

Entretanto, é importante observar que, dada a natureza do Naive Bayes, ele terá um baixo desempenho em datasets desbalanceados que é o caso do Avila.

### Modelo Padrão

Primeiro treinaremos o modelo Gaussian com os valores padrões para determinar uma base que será utilizada a fim de comparar resultados e buscar melhorias.

### Análise do modelo

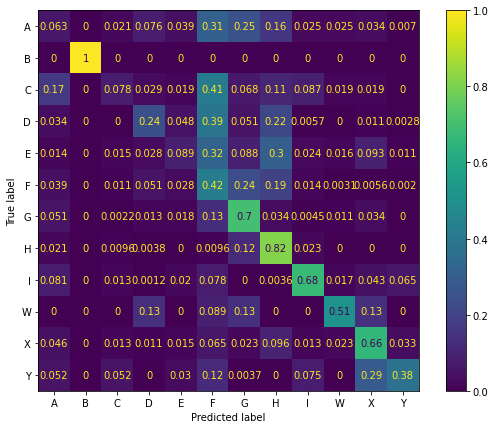


No geral nota-se o baixo desempenho do modelo em relação a todos os scores (Precision, Recall e F1-score).   
 Além disso, nota-se uma grande disparidade entre os scores de cada classe.

* Para a classe A o precision é de 0.51 enquanto o recall e f1-score são muito menores, 0.06 e 0.11 respectivamente.
* Classes como I e B tiveram scores altos em relação à média dos scores das classes, mas para classes como a C desempenharam mal.

#### Matriz de Confusão

A matriz de confusão a seguir demonstra como o modelo se comportou tentando prever o conjunto de testes.



Nota-se uma dificuldade do modelo em predizer corretamente algumas classes

* A classe A é frequentemente confundida com a F e a G.
* A classe C é frequentemente confundida com a F.

Enquanto teve uma acurácia relativamente melhor em outras, à exemplo das classes H e I.

Adjunto, o modelo tende a prever frequentemente um exemplo como sendo pertencente à classe F.

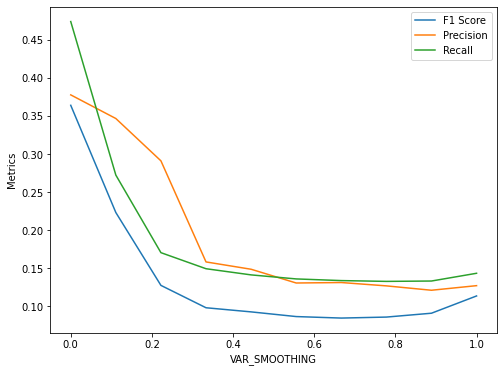
### Fine-Tunning

Para realizar o fine-tunning do modelo iremos tentar achar os melhores hiperparâmetros cujo o desempenho dos scores ( Precision, F1, Recall) seja máximo.

O Gaussian apenas possui um hiperparâmetro que é **var\_smoothing**.

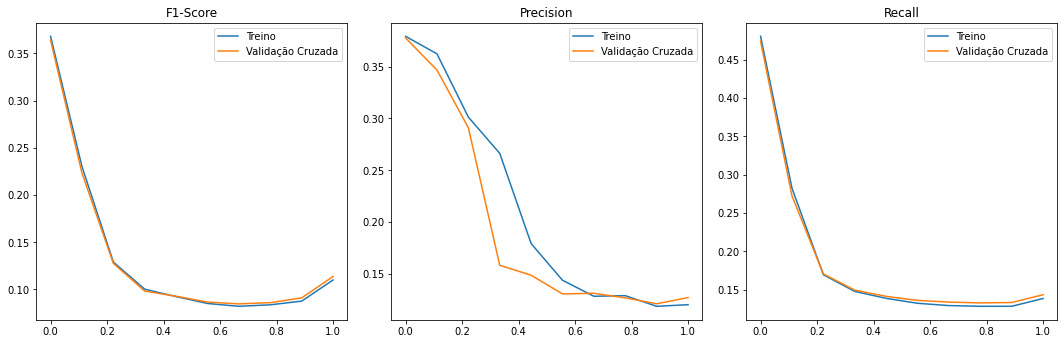
O var\_smoothing é um valor entre 0 e 1 e indica a parte da maior variância de todas as features que são adicionadas às variâncias a fim de estabilidade de cálculo. Possui valor padrão de 1e-9.

Para fazer a busca do melhor var\_smoothing é feito um grid\_search, que utiliza dentro de seu processo a validação cruzada. Assim evitamos o bias de tunning com dataset de teste.



É possível notar que todos os scores seguem a mesma tendência de piora inicial conforme o aumento do var\_smoothing.

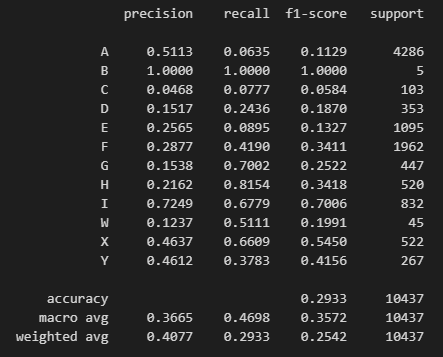
É válido ressaltar que existe uma queda brusca para var\_smoothing entre 0.0 e 0.2, seguida por uma estabilização para valores entre 0.2 e 0.9, para enfim ocorrer uma crescente de 0.9 a 1.0.



Observando os gráficos vemos que não ocorre overfitting. Além disso, observamos também que quanto mais próximo de 0 é o var\_smoothing, maior são os scores.

### Modelo Otimizado

Assim, para realizar o fine-tunning do modelo selecionaremos 1e-12 já que esse é o melhor valor do processo de grid search.



É possível notar a melhora, apesar de mínima, em todos os scores quando comparado com o modelo padrão.

É válido ressaltar que o naive bayes não terá no geral um bom desempenho pois o dataset está desbalanceado.

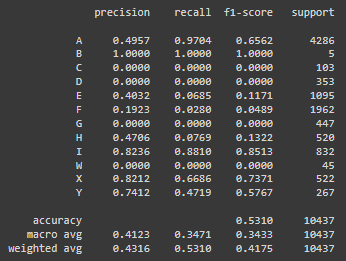
## Classificação usando Regressão Logística

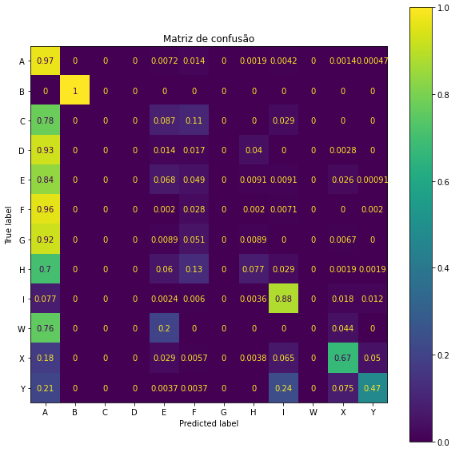
O primeiro ponto importante a ser notado é que o problema em questão é de classificação multi-classe. O algoritmo de regressão logística consegue separar um dado entre 2 classes, tal como um problema de sim ou não. Assim, para um problema multi-classe, podemos tratar cada classe como um problema de classificação binária, ou seja, vamos comparar, para cada classe, se um dado pertence a ela ou não. Essa abordagem é conhecida como "método de um contra todos". Nessa abordagem, ao trabalharmos com uma classe, essa classe é denotada por 1 e as outras, por 0.

### Modelo base

Parâmetros:

* multi\_class='ovr
* solver='lbfgs'





#### Conclusão

Não tivemos um bom resultado. Para buscar os melhores parâmetros, vamos usar um grid search com cross validation.

### Grid search

Parâmetros testados:

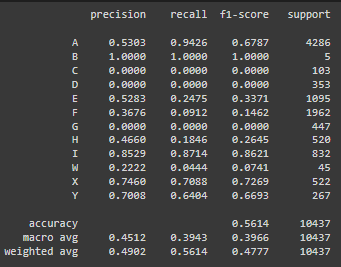
* multi\_class: ["ovr", "multinomial"]
* solver: ['lbfgs', 'newton-cg', 'sag', 'saga']

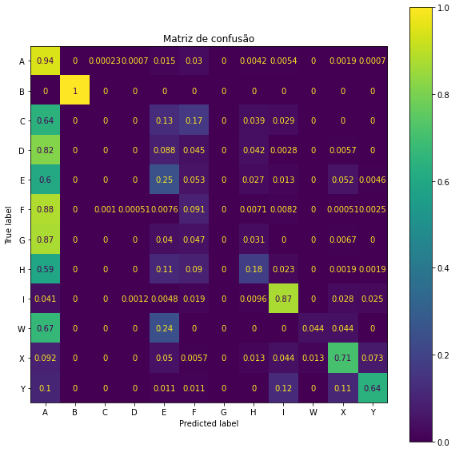
#### Resultado:



Analisando a tabela, é possível identificar que os melhores parâmetros são "multinomial" para "multi\_class" e "lbfgs" para "solver. A partir disso, todos os outros modelos que treinarmos usarão esses parâmetros.

### Modelo com melhores parâmetros



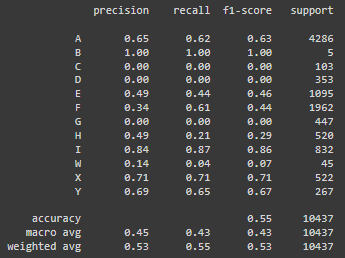


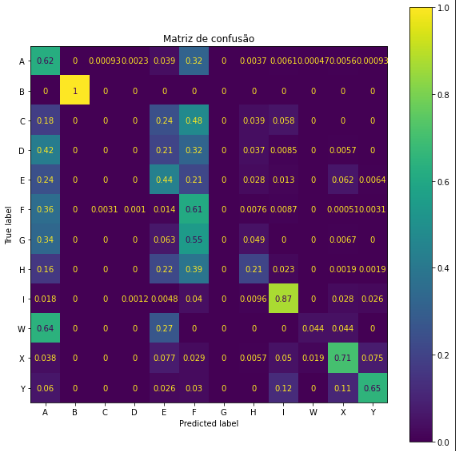
#### Conclusão:

Foi possível perceber que o modelo base errou todos das classes C, D e G, além de ter errado a maioria dos exemplos das classes E, F, H e W. Além disso, um fato muito interessante de se notar é que todas essas classes que não tiveram bons resultados foram confundidas com exemplos da classe A. Isso provavelmente aconteceu devido ao desbalanceamento do dataset, com muito mais dados da classe A. Assim, para a primeira tentativa de melhoria do modelo, vamos apenas diminuir a quantidade de exemplos da classe A e manter os outros parâmetros.

### Modelo com mudança no balanceamento do dataset

Para este modelo, vamos reduzir o número de exemplos da classe A em 50% e conferir seus resultados.





#### Conclusão

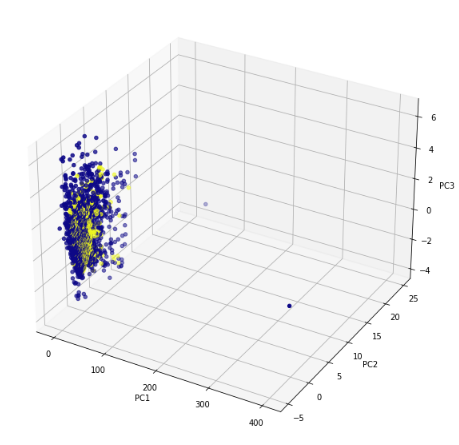
O resultado da classificação melhorou muito para algumas das classes, como a E e a F, mas piorou para a classe A. Observando as matrizes de confusão dos dois modelos testados até aqui, é possível inferir que as classes A e F estão gerando muitos problemas para a classificação. Isso pode estar acontecendo porque ambas as classes devem possuir exemplos muito próximos uns dos outros, dificultando sua separação. Vamos utilizar a técnica do PCA, que permite a redução da dimensionalidade do dataset, para identificar as features que melhor separam ambas as classes e utilizá-las para realizar a classificação.

### Modelo com mudança nas colunas do dataset

Inicialmente vamos tentar procurar formas de separar as classes A e F, a fim de melhorar a acurácia do modelo. Para isso, vamos tentar utilizar o PCA.

#### Análise utilizando PCA

Reduzimos o número de features para 3. Com o PCA feito, podemos plotar um gráfico para verificar se os componentes principais conseguem separar as classes A e F.



Utilizando o PCA foi possível ver que, mesmo utilizando os principais componentes para separar os dados, eles ainda não são bem separáveis. Assim, vamos tentar reduzir o número de features, mantendo as mais relevantes de acordo com o artigo que acompanha o projeto.

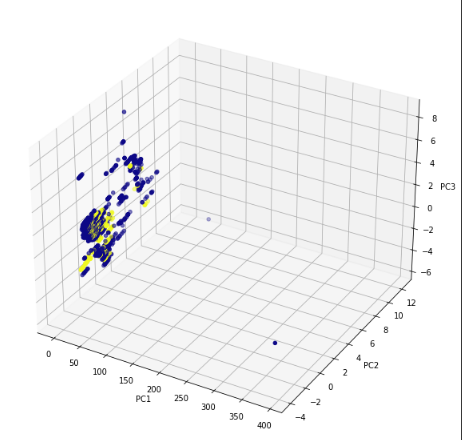
#### Removendo colunas do dataset

Removemos as colunas:

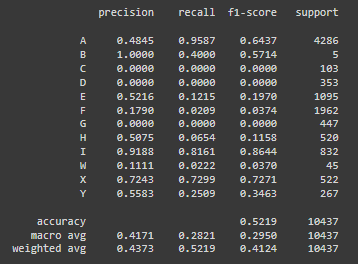
* modular ratio/interlinear spacing
* weight
* modular ratio
* interlinear spacing

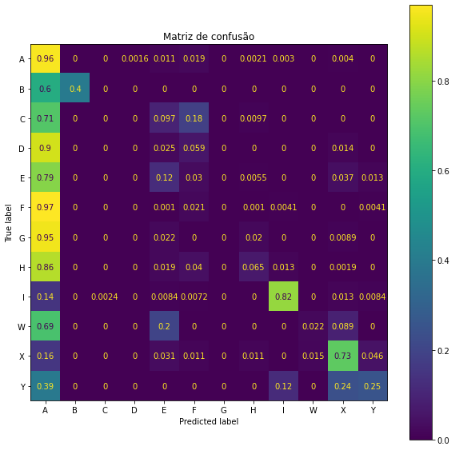
#### Análise utilizando PCA para dataset com colunas removidas

Reduzimos o número de features para 3 novamente.



Mesmo com as features menos relevantes removidas, ainda há indecisão na separação, mas vamos testar dessa forma.





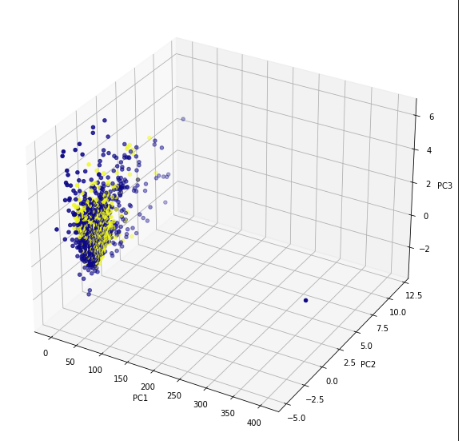
#### Conclusão

Não foi possível obter melhorias no modelo removendo algumas colunas, principalmente devido ao desbalanceamento do dataset. Assim, vamos tentar mesclar as abordagens: reduzir o número de casos e remover features menos importantes.

### Modelo com mudança nas colunas e no balanceamento do dataset

Para este modelo, vamos reduzir o número de exemplos das classes A e F e remover as colunas menos relevantes para a classificação.

#### Análise utilizando PCA



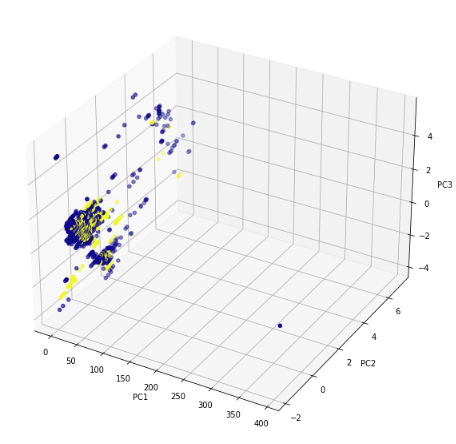
Utilizando o PCA foi possível ver que, mesmo utilizando os principais componentes para separar os dados, eles ainda não são bem separáveis. Assim, vamos tentar reduzir o número de features, mantendo as mais relevantes de acordo com o artigo que acompanha o projeto.

#### Removendo colunas do dataset

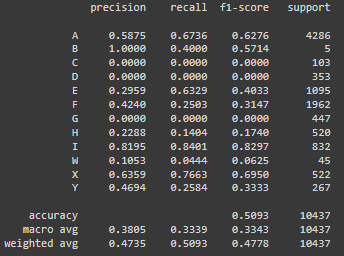
Removemos as colunas:

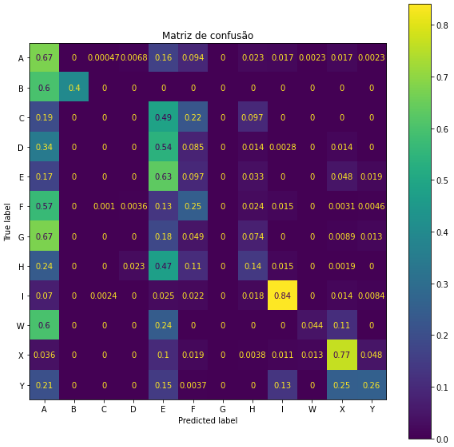
* modular ratio/interlinear spacing
* weight
* modular ratio
* interlinear spacing

#### Análise utilizando PCA para dataset com colunas removidas



Dessa vez é possível ver um grau um pouco maior de separação, principalmente se considerarmos os exemplos que estão mais distantes do bloco que concentra a maioria dos exemplos.



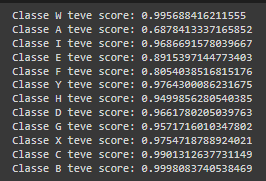


#### Conclusão

Mesmo com as mudanças, não foi possível melhorar a acurácia. Além disso, com menos exemplos no treinamento, o modelo não conseguiu informações suficientes para classificar A e F, o que também contribuiu para piorar a acurácia.

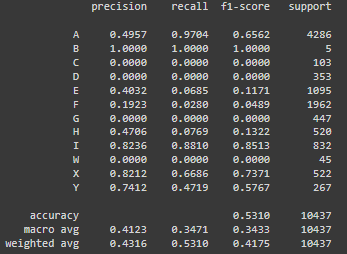
### Abordagem One vs Rest Manual

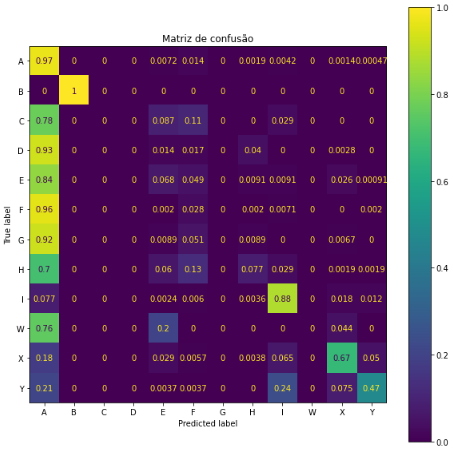
#### Scores individuais



Pelos scores, é possível ver que, ao menos a nível individual, os modelos conseguem diferenciar entre exemplos que pertencem às classes e exemplos que não pertencem, realizando uma avaliação binária.

#### Resultados





#### Conclusão

Ainda que os modelos performem bem em avaliações binárias, o resultado permanece ruim quando eles são unidos. Isso pode ter ligação com a forma escolhida para uni-los, já que ela foi feita escolhendo como classe final aquela que possui maior probabilidade. Além disso, pelos scores individuais, é possível perceber que as classes que possuem mais exemplos (A, E e F) são as que têm menor score, o que indica que o número de casos pode estar introduzindo um viés na classificação dos exemplos.

### Grid search com pesos

O grid search com pesos para cada classe não foi feito antes pois, como fizemos mudanças no dataset, seria necessário procurar pesos para cada mudança específica.

Parâmetros testados:

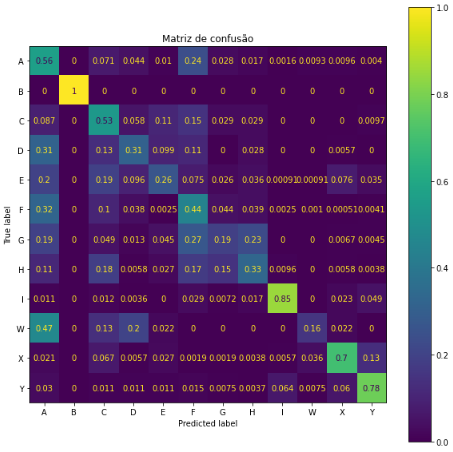
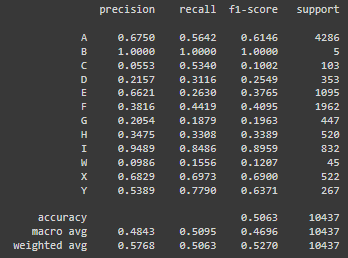
* multi\_class: ["ovr", "multinomial"]
* solver: ['lbfgs', 'newton-cg', 'sag', 'saga']
* class\_weights:
  + [{'A': 1, 'B': 857, 'C': 219, 'D': 104, 'E': 21, 'F': 9, 'G': 104, 'H': 89, 'I': 53, 'W': 392, 'X': 70, 'Y': 166},
  + {'A': 0.1, 'B': 85.7, 'C': 21.9, 'D': 10.4, 'E': 2.1, 'F': 0.9, 'G': 10.4, 'H': 8.9, 'I': 5.3, 'W': 39.2, 'X': 7.0, 'Y': 16.6},
  + {'A': 1, 'B': 6, 'C': 5, 'D': 4, 'E': 2, 'F': 1, 'G': 4, 'H': 4, 'I': 3, 'W': 5, 'X': 4, 'Y': 5},
  + {'A': 1, 'B': 1, 'C': 10, 'D': 10, 'E': 5, 'F': 4, 'G': 10, 'H': 10, 'I': 1, 'W': 10, 'X': 4, 'Y': 5},
  + {'A': 1, 'B': 1, 'C': 40, 'D': 12, 'E': 3, 'F': 2, 'G': 10, 'H': 8, 'I': 2, 'W': 10, 'X': 4, 'Y': 10},
  + {'A': 1.5, 'B': 1, 'C': 50, 'D': 12, 'E': 3, 'F': 3, 'G': 8, 'H': 6, 'I': 2, 'W': 10, 'X': 4, 'Y': 10}]

No total foram 48 possibilidades, de modo que não é possível colocar uma imagem da tabela neste documento.

Melhores parâmetros:

* multi\_class = 'multinomial'
* solver = 'newton-cg'
* class\_weight = {'A': 1.5, 'B': 1, 'C': 50, 'D': 12, 'E': 3, 'F': 3, 'G': 8, 'H': 6, 'I': 2, 'W': 10, 'X': 4, 'Y': 10}

### Modelo com os melhores pesos



#### Conclusão

Este foi o modelo com melhor F1-score dentre todos os testados. Ainda assim, não foi possível encontrar um modelo com resultados satisfatórios.

### Conclusão geral

Ainda que os modelos performem bem em avaliações binárias, o resultado permanece ruim quando eles são unidos. Isso pode ter ligação com a forma escolhida para uni-los, já que ela foi feita escolhendo como classe final aquela que possui maior probabilidade. Além disso, pelos scores individuais, é possível perceber que as classes que possuem mais exemplos (A, E e F) são as que têm menor score, o que indica que o número de casos pode estar introduzindo um viés na classificação dos exemplos.

## Classificação usando K-Nearest Neighbors (KNN)

O KNN é um modelo de aprendizado de máquina que se baseia na distância entre os elementos para realizar sua classificação. Sendo assim, esse modelo tem uma melhor performance em datasets nos quais as classes estão bem separadas. Contudo, como visto no PCA realizado anteriormente, existem classes não bem separadas, o que pode afetar o desempenho do modelo.

Fixaremos a distância euclidiana e tentaremos encontrar um valor de K que maximize as métricas em análise.

### Modelo Base

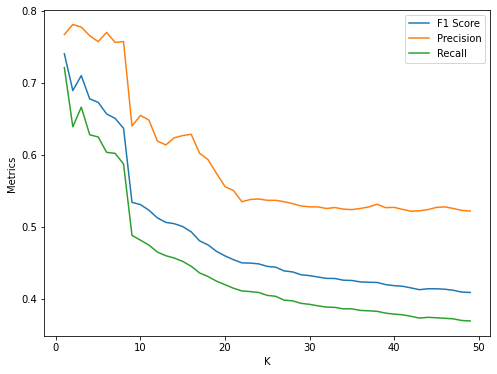
De início, treinamos um modelo mais básico com k=3 para que possamos perceber o ganho de desempenho ocasionado por uma seleção mais cuidadosa do parâmetro. A acurácia do treino foi de 87% e a do teste, 76%.

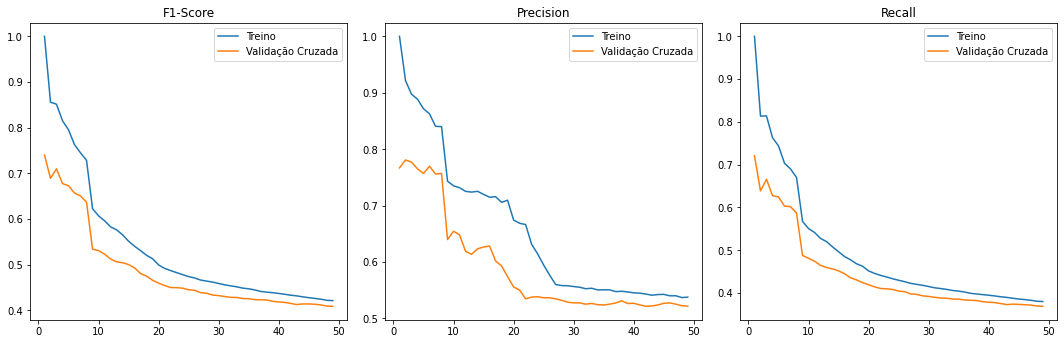
Como é esperado que a acurácia não caia para abaixo do modelo base, percebe-se que o KNN se sai relativamente bem na tarefa dada embora hajam classes não bem definidas

### Escolhendo os Melhores Parâmetros

Para alcançar melhores resultados, buscaremos o valor de K que traz os melhores resultados para o modelo. Exploraremos os resultados de F1-Score, Precision e Recall de modelos sobre um conjunto de valores entre 1 e 49. Para essa tarefa, usaremos o **GridSearch** e passaremos ao parâmetro **n\_neighbors** o intervalo de valores de K a serem explorados.

### Análise

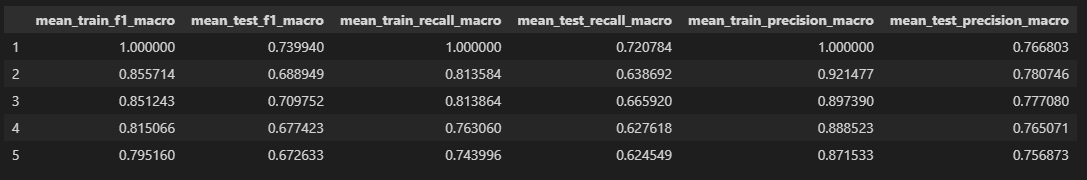




Pelo gráfico acima, percebe-se que todas as métricas se comportam de maneira bastante similar ao longo dos valores de K analisados.

Percebe-se que os melhores valores de K estão mais à esquerda, sendo aqueles com menor magnitude. É válido ressaltar a queda brusca em todas as métricas que há no ponto onde K=8.

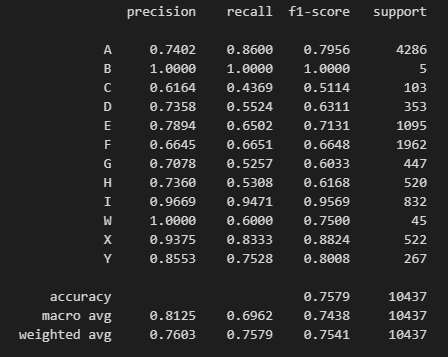
Como pode ser visto, ainda no gráfico, os pontos com as melhores métricas podem ser obtidos entre os valores de K 1 e 5, logo, exploraremos seus valores.

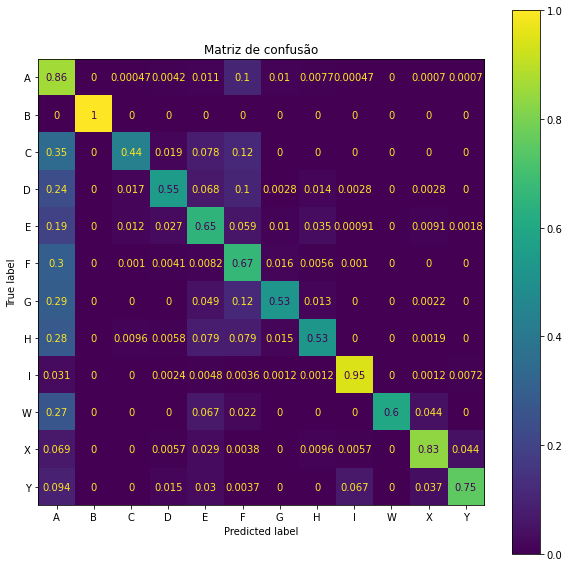


Embora o K=1 apresente as melhores métricas, é perceptível que ocorreu overfitting nele uma vez que as métricas do treino estão todas perfeitas. Além disso, um KNN equipado com K=1 é um modelo instável. Por esse motivos, não escolhemos o melhor valor de K como sendo 1. Sendo assim, o melhor valor de K para o modelo é 3 que, por coincidência, é o mesmo que testamos no modelo base.

### Análise

Avaliamos abaixo o desempenho do modelo encontrado na classificação usando as métricas F1-Score, Precision, Recall e Acurácia. Além disso, usaremos a matriz de confusão para facilitar o entendimento de como as classes estão sendo classificadas.



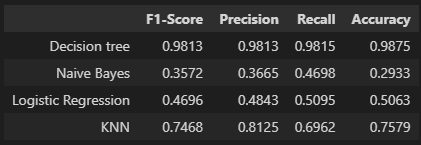


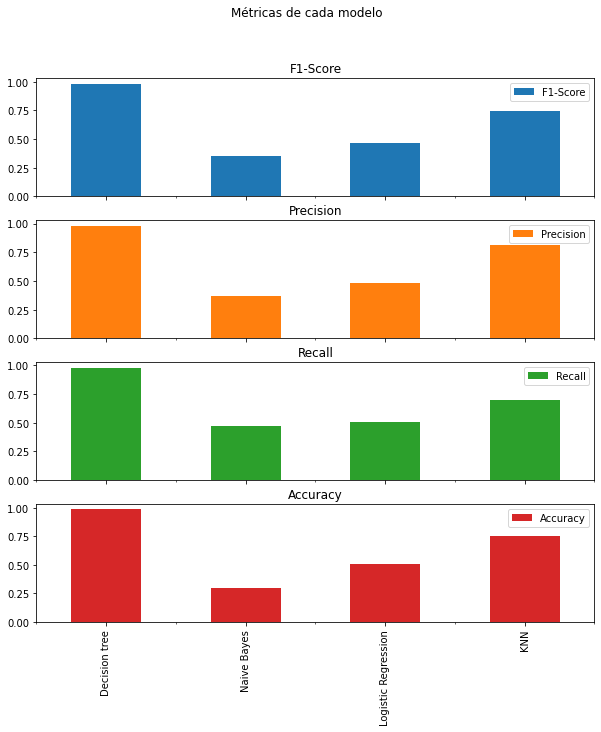
Percebe-se que o KNN obteve uma performance razoável em solucionar o problema. Como pode ser visto na matriz de confusão, o modelo constantemente prediz a classe A, mesmo quando ela não é a correta, provavelmente pelo fato dessa classe possuir a maior quantidade de exemplos.

Nota-se também que o modelo se sai melhor em classificar algumas classes do que outras. A classe B tem seus exemplos do teste perfeitamente classificados, enquanto que a o modelo falha em classificar exemplos de C mais do que metade das vezes.

## Comparação dos Modelos

Com base nas avaliações realizadas acima, podemos gerar uma tabela e um gráfico com o desempenho dos modelos avaliados sobre as métricas escolhidas





É perceptível que o modelo usando Árvores de Decisão supera todos os demais modelos nas métricas dadas pelo fato do mesmo não assumir uma distribuição dos dados.

O KNN ocupou o segundo lugar, obtendo métricas razoáveis embora haja classes muito semelhantes.

A Regressão Logística e o Naive Bayes ficaram em 3º e 4º lugar, devido ao desbalanceamento do dataset e a semelhança entre as classes.

## Anexos

Repositório do Github:

## Referências

C. De Stefano, M. Maniaci, F. Fontanella, A. Scotto di Freca,

Reliable writer identification in medieval manuscripts through page layout features: The “Avila” Bible case, Engineering Applications of Artificial Intelligence, Volume 72, 2018, Pages 99-110,ISSN 0952-1976,

https://doi.org/10.1016/j.engappai.2018.03.023.

scikit-learn.org/stable/modules/cross\_validation.html

scikit-learn.org/stable/modules/grid\_search.html#grid-search

scikit-learn.org/stable/modules/model\_evaluation.html#multimetric-scoring

scikit-learn.org/stable/auto\_examples/model\_selection/plot\_multi\_metric\_evaluation.html#sphx-glr-auto-examples-model-selection-plot-multi-metric-evaluation-py

towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f#989c

satishgunjal.com/multiclass\_lr\_sklearn/#2020

machinelearningmastery.com/multinomial-logistic-regression-with-python/

scikit-learn.org/stable/modules/generated/sklearn.linear\_model.LogisticRegression.html

builtin.com/machine-learning/logistic-regression-classification-algorithm

www.statology.org/pandas-drop-rows-with-condition/

www.geeksforgeeks.org/implementing-pca-in-python-with-scikit-learn/

towardsdatascience.com/the-perfect-recipe-for-classification-using-logistic-regression-f8648e267592

<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>

<https://scikit-learn.org/stable/modules/tree.html#minimal-cost-complexity-pruning>

<https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html>

<https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html>

<https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html>