Aprendizagem de Máquina

SVM Classificando base de dados Glass

Marcos de Souza Oliveira (mso2@cin.ufpe.br)

Carlos Antonio Alves Junior (caaj@cin.ufpe.br)

Matheus Johann Araújo (mja@cin.ufpe.br)

20/07/2021

Missão:

A missão 5 consistiu na tarefa de realizar uma comparação de diferentes configurações de SVM na base de dados *Glass*. Para tal comparação, foi necessário conduzir uma análise exploratória dos dados da base *Glass* e posteriormente fazer o pré-processamento dos mesmos. Apresentamos abaixo a sequência de análise e pré-processamento, bem como os resultados da criação dos modelos para classificação dos exemplos dispostos na base *Glass*.

Antes de realizarmos o processo de construção (treinamento) final do modelo, exploramos de forma sistemática os melhores hiperparâmetros para cada função de kernel que foi utilizada.

Decorrido o exposto acima, é demonstrado o resultado da comparação dos diferentes modelos aprendidos usando abordagens de funções kernels distintas.

O código de todo o projeto se encontra disponibilizado no GitHub (https://github.com/matheusjohannaraujo/missao_5). Para que fosse possível o desenvolvimento do projeto de forma colaborativa, optamos por utilizar o Google Colaboratory, pois permite execução de código python, e provê um meio de observar os resultados pelo navegador, o link abaixo permite acessar o projeto. https://drive.google.com/file/d/1mKFbpwyZALDN0apSjVOmPCCywSm0B6Ab/

Análise e processamento dos dados:

A base de dados *Glass* contém 214 instâncias. Cada instância descreve diferentes tipos de vidros em termos de seus componentes de oxidação (Ferro, Magnésio, etc...). No total, existem 6 tipos de vidros (classes), e a distribuição de cada classe pode ser vista no gráfico abaixo (Figura 1).

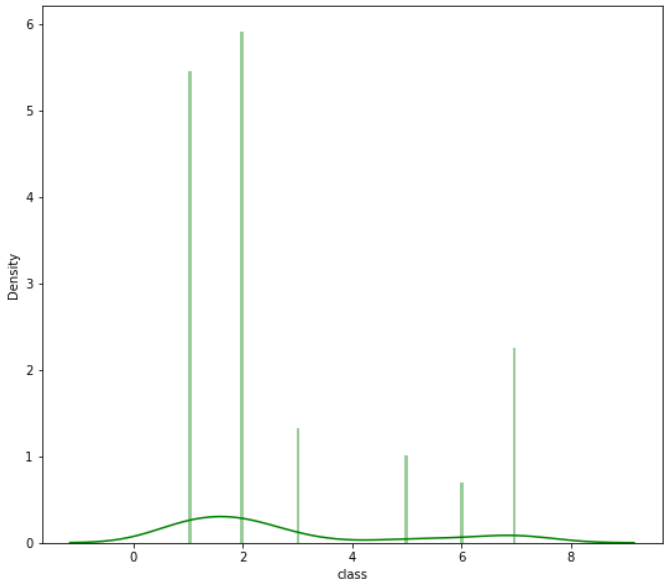


Figura 1. Gráfico de distribuição de cada classe
ao longo das 214 instâncias de vidros.

Nota-se um grande “*spike*” nas classes 1 e 2, além de uma distribuição não contínua (o valor 4 é pulado). Para resolver o primeiro problema, é necessário fazer uma divisão estratificada dos dados na separação para teste, validação e treino. Já no problema da não continuidade, é necessário apenas um mapeamento para uma distribuição contínua. A figura 2 mostra a distribuição após o mapeamento.

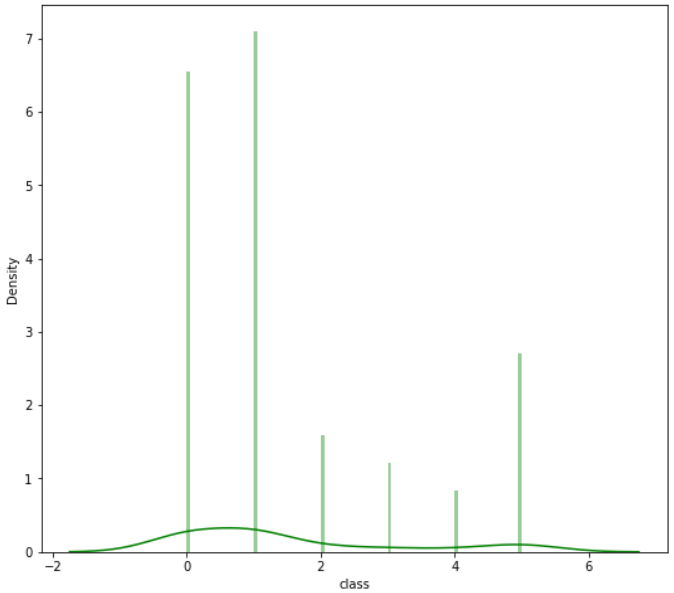


Figura 2. Gráfico de distribuição de
cada classe após mapeamento.

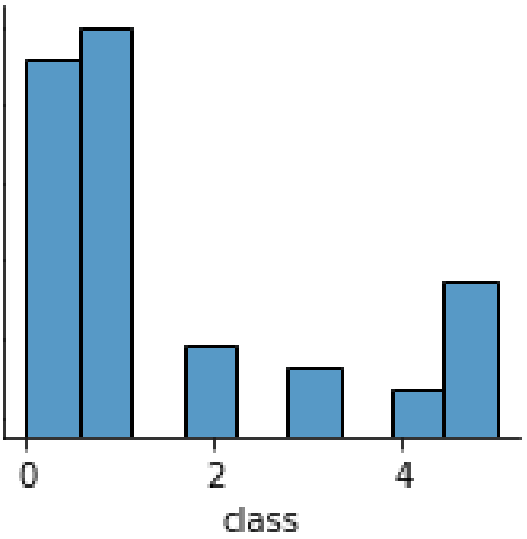
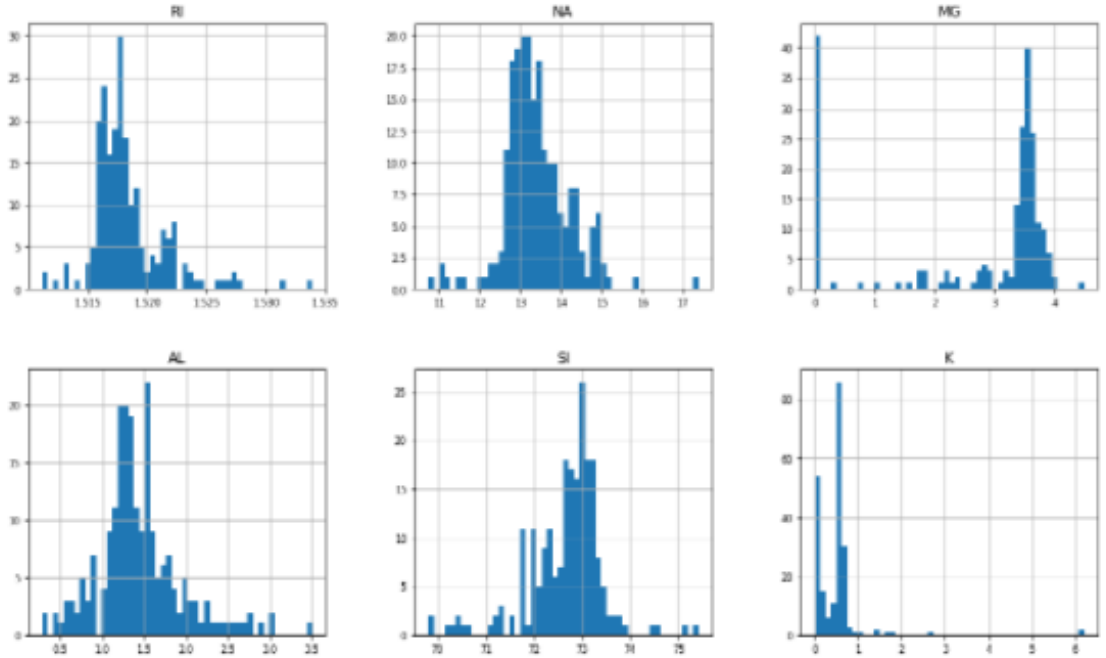


Figura 2.1. Gráfico que demonstra de forma mais visível a distribuição das classes.

Após a análise das classes, partimos para a análise dos atributos. Ao todo, são 10 atributos: “ID”, “RI”, “NA”, “MG”, “AL”, “SI”, “K”, “CA”, “BA”, “FE”, com exceção do atributo “ID”, todos dizem respeito à oxidação do vidro.

Uma forma de visualizar o comportamento de cada atributo, é através do histograma. Na figura 3 é possível ver o histograma de cada atributo no dataset.



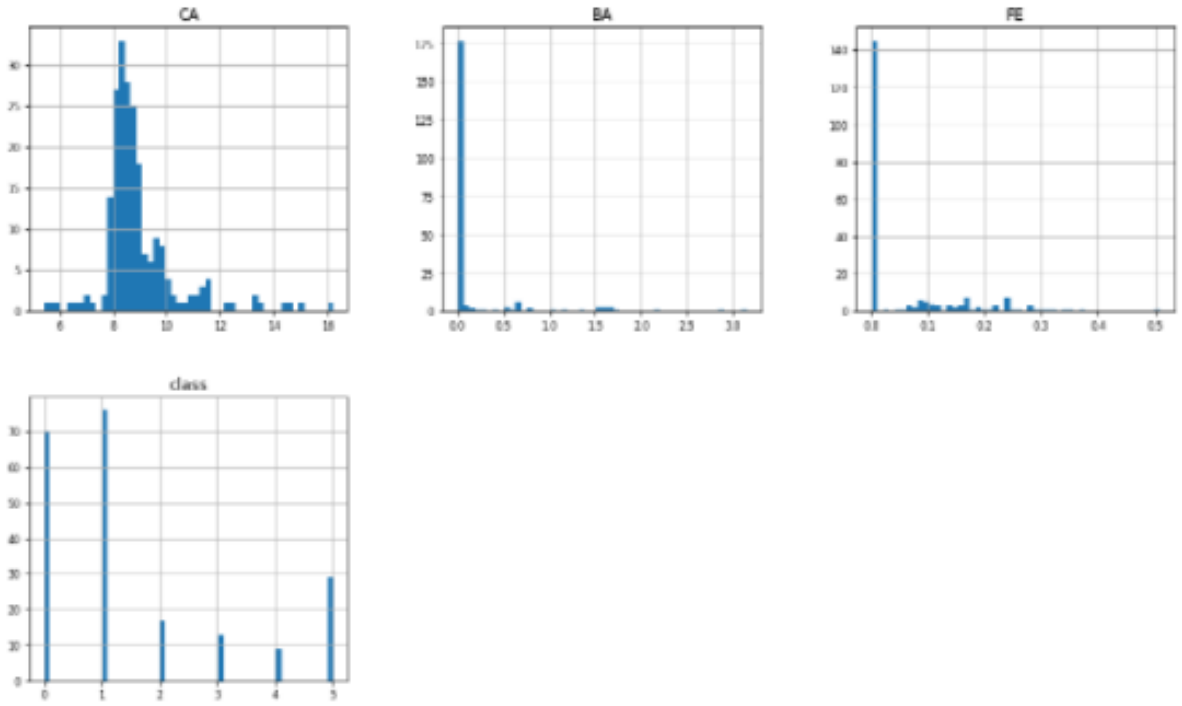
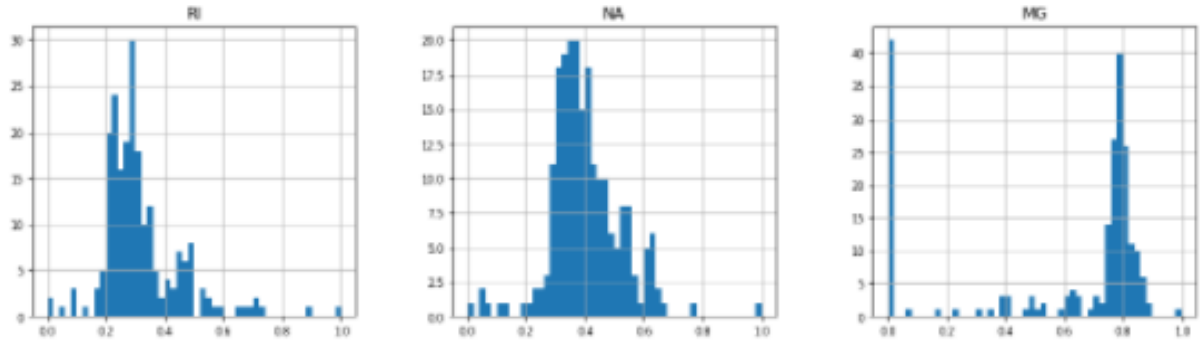


Figura 3. Histograma de cada atributo (incluindo a classe).

Pela análise dos histogramas, é possível perceber que os dados estão em intervalos de valor muito distintos (um exemplo claro é o do atributo “FE” comparado com o SI, o primeiro tem seus valores no intervalo de 0 até 3, enquanto o outro, 70 a 75 majoritariamente). Decidiu-se então, fazer a normalização dos atributos, afim de tentar amenizar a diferença entre os intervalos. A figura 4 mostra os atributos após a normalização.



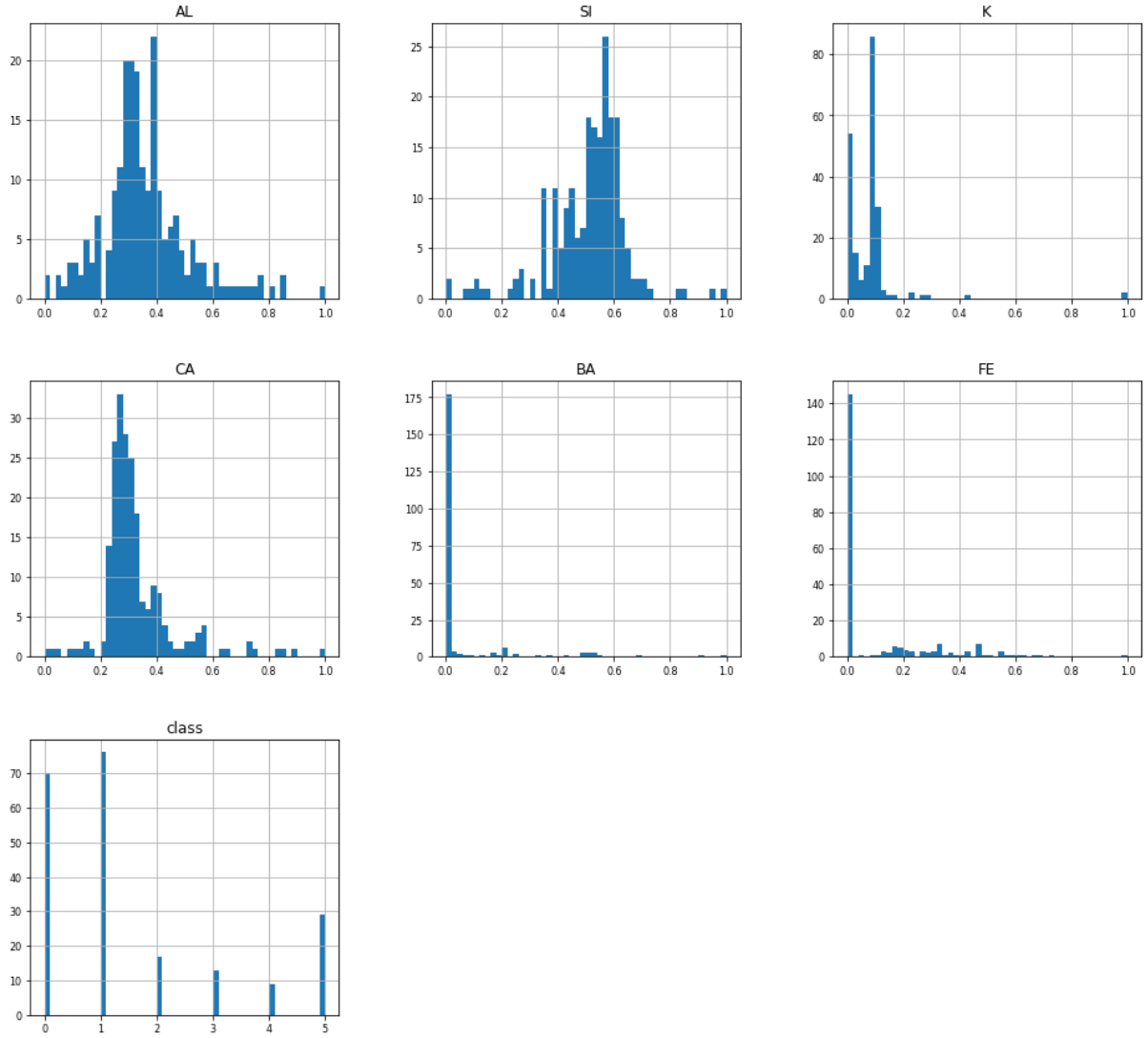


Figura 4. Histograma dos atributos após a normalização.

Após a normalização, foram feitas análises sobre a correlação entre os atributos. Inicialmente, buscamos os atributos que estavam mais relacionados com a classe, e 4 dos 9 atributos (desconsiderando o ID) estavam fortemente correlacionados com a classe. A tabela 1 mostra a relação entre eles.

Atributo	Correlação
AL	0.591198
BA	0.577676
NA	0.506424

MG	-0.728160

Tabela 1. Correlação entre atributos e classe.

Após a análise, decidiu-se testar os modelos com a inclusão desses atributos (mesmo com o alto nível de correlação). Foi aplicado também o algoritmo do PCA, para que as correlações fossem resolvidas, gerando um dataset mais conciso e com menos correlações entre os atributos.

Além disso, geramos um mapa de correlação entre os atributos (Figura 5), de forma a explicitar as correlações além das classes.

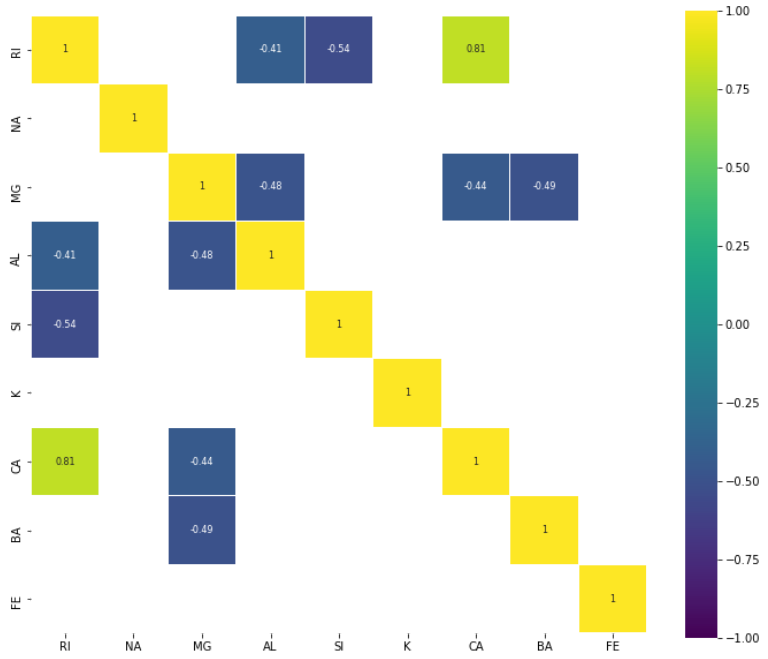


Figura 5. Matriz de correlação entre os atributos.

A análise e o processamento foram realizados tendo como base o seguinte artigo: [Detailed exploratory data analysis with python | Kaggle].

Resultados e Comparação:

Ambiente:

Após a análise e processamento dos dados, partimos para a aplicação do método.

Foi utilizada a biblioteca *Scikit-learn*, mais especificamente, a função SVM.svc. Os quatro kernels utilizados foram: Polynomial, RBF, Sigmoid e Linear. Antes de realizar o treinamento do modelo, foram selecionados os melhores hiperparâmetros para cada Kernel.

Além disso, as bibliotecas *numpy*, pandas, *mathplotlib* e *seaborn* também foram utilizadas na manipulação de dados e *plot* de gráficos.

Tudo foi feito no *Google Collab*, utilizando a linguagem de programação Python.

Separação dos dados:

Foi feita a divisão de dados para treino e validação (k-fold) e teste. A divisão foi feita de forma a separar 20% dos dados para testes e 80% para o treino/validação através do K-fold.

O K-fold foi do tipo 5x10. A divisão dos dados foi feita de forma estratificada em todas as etapas.

Resultados e discussão:

Oito classificadores foram treinados simultaneamente (4 kernels com os atributos fortemente correlacionados e 4 sem os atributos), todos com os melhores hiperparâmetros para cada função de kernel.

Os classificadores cujos atributos fortemente correlacionados, não foram retirados, obtiveram uma melhor performance no conjunto de treino do que os classificadores treinados sem os atributos fortemente correlacionados. Esse resultado não se repetiu completamente no conjunto de testes, onde metade dos

classificadores (sem atributos fortemente correlacionados) obtiveram performance melhor ou igual aos classificadores com atributos fortemente correlacionados. Uma comparação gráfica (da acurácia durante o treino) pode ser vista nas figuras abaixo:

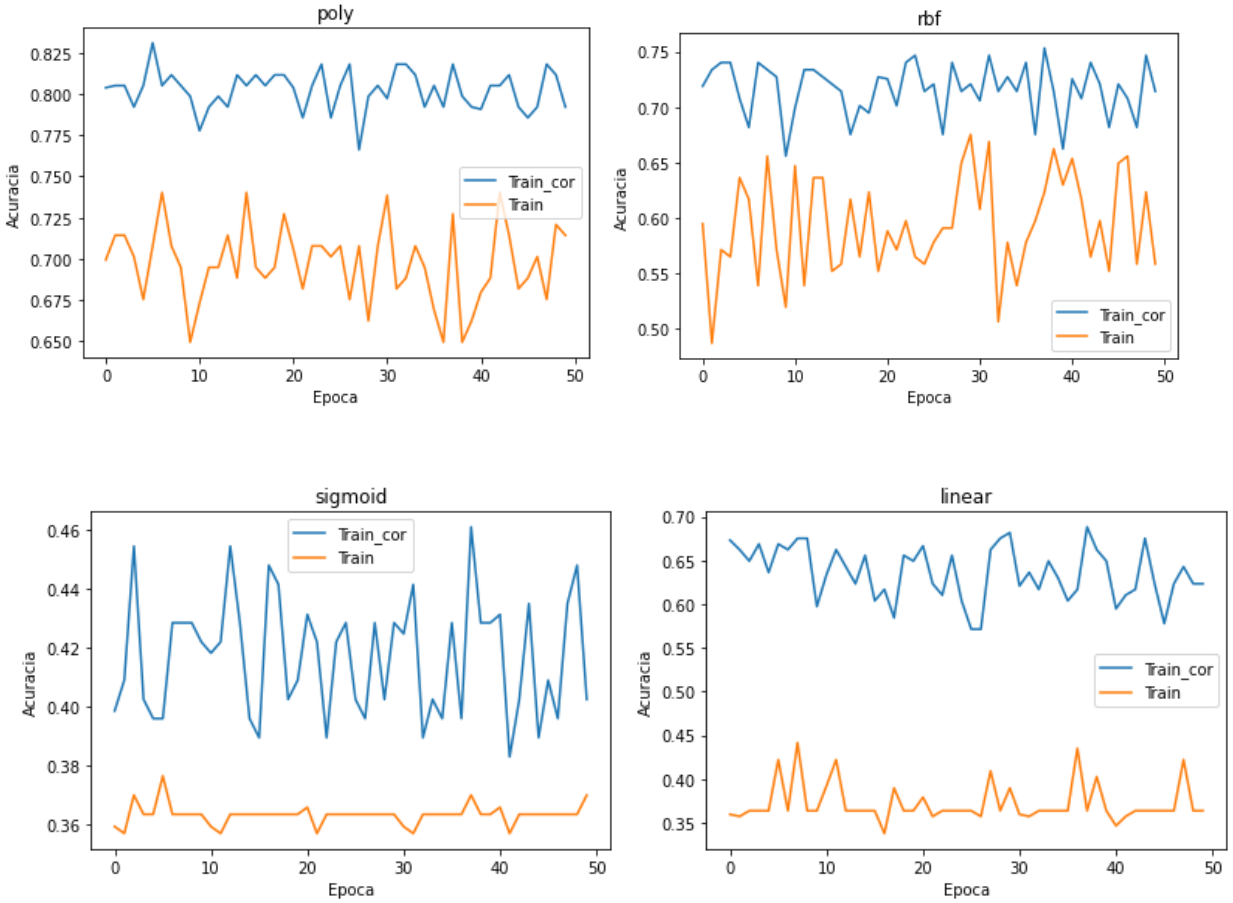


Figura 6. Diferença entre os classificadores treinados com e sem as features fortemente correlacionadas.
(Época diz respeito ao 5, do 5xK-Fold)

Os resultados da aplicação no conjunto de testes pode ser vistos no relatório abaixo:

```
------------------------------------------------------------
Acurácia do conjunto de testes:
------------------------------------------------------------
Kernel POLY
Score: 0.6744186046511628
Score (sem atributos correlacionados): 0.6744186046511628

Kernel RBF
Score: 0.627906976744186
Score (sem atributos correlacionados): 0.6744186046511628

Kernel SIGMOID
Score: 0.5581395348837209
Score (sem atributos correlacionados): 0.5348837209302325

Kernel LINEAR
Score: 0.5813953488372093
Score (sem atributos correlacionados): 0.5348837209302325
```

Relatório 1. Acurácia dos classificadores no conjunto de testes. As SVMs polynomial e RBF sem atributos correlacionados se saíram melhor/iguais às suas contrapartes.

Além disso, é possível ver claramente que as SVMs com o kernel polynomial e RBF performam melhor no geral. Sendo o kernel polynomial, o melhor. As figuras abaixo mostra um gráfico comparando a performance dos diferentes kernel ao longo do treino e da validação, acentuando ainda mais a superioridade do kernel polynomial:

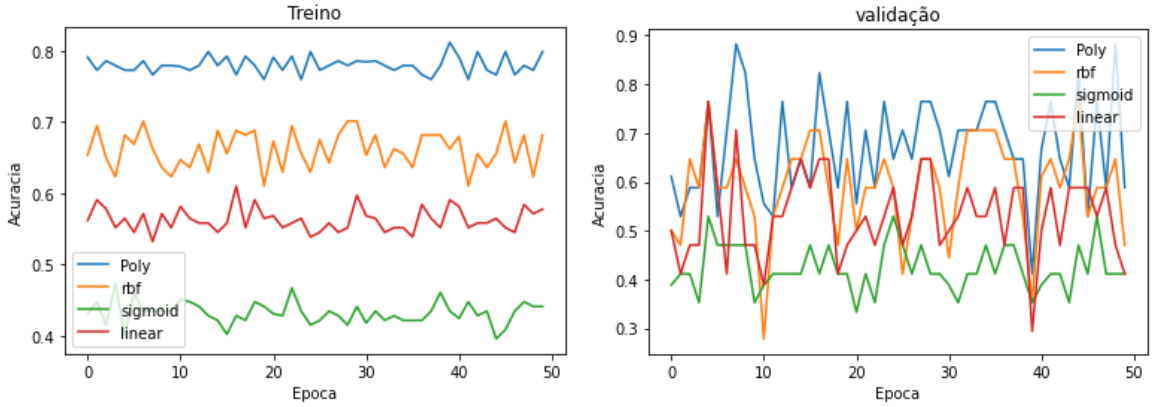


Figura 7. Comparação entre os diferentes kernels (Com atributos fortemente correlacionados).

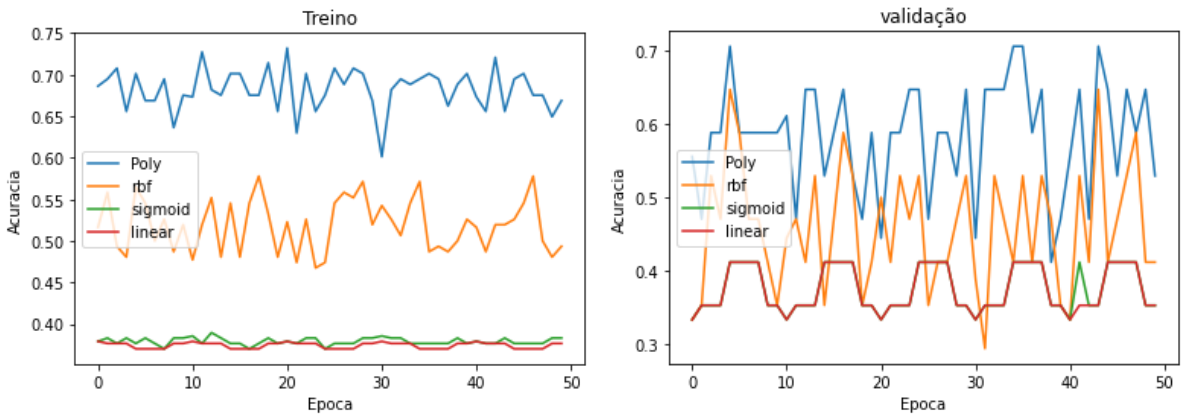


Figura 8. Comparação entre os diferentes kernels
(Sem atributos fortemente correlacionados).

Por fim, plotamos as matrizes de confusão para cada kernel:

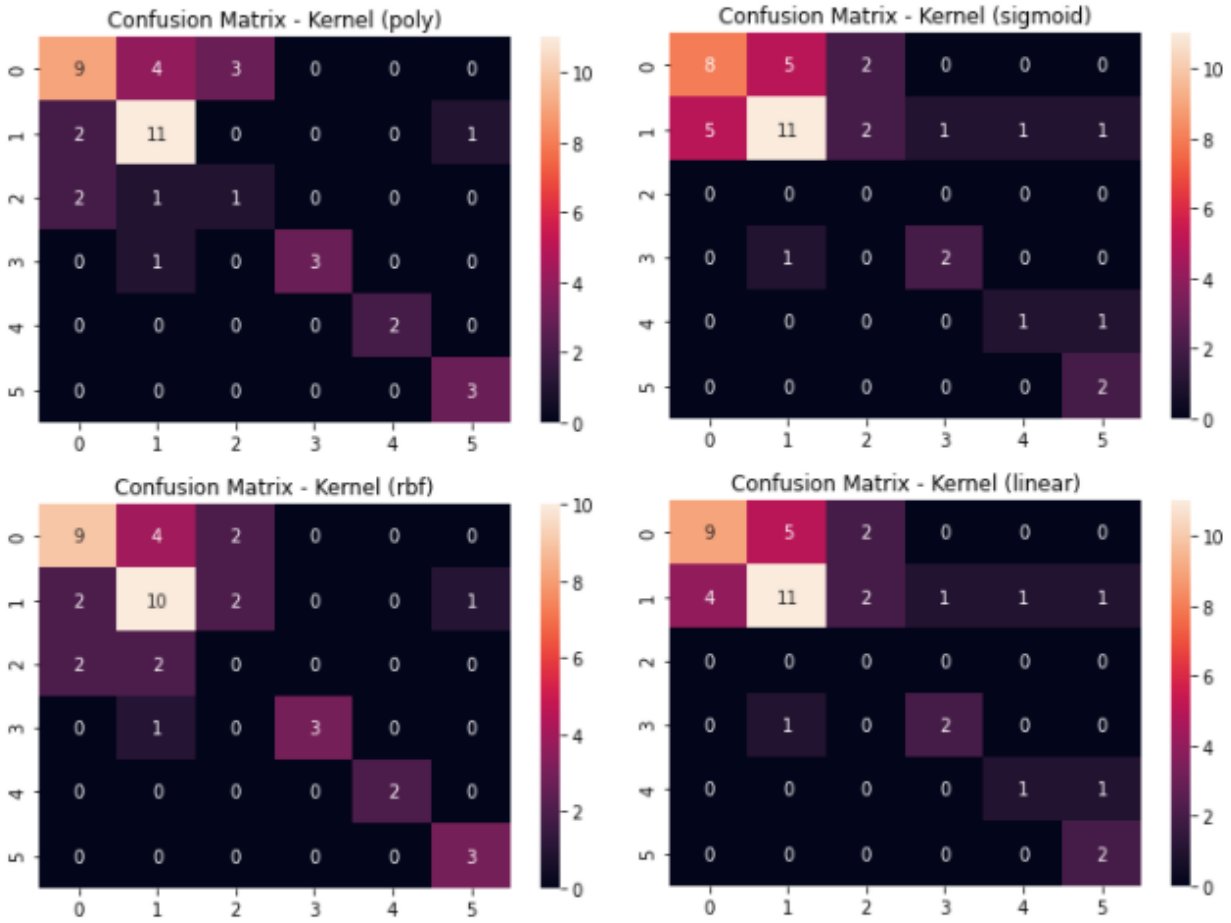


Figura 9: As matrizes de confusão obtidas no SVM
utilizando funções de Kernel diferentes.

Percebe-se que, em todos os casos, as classificações com as maiores quantidades de erros, são justamente as classes mais volumosas (0 e 1). Isso é o resultado da maior quantidade de exemplos nessas classes, problema que, mesmo remediado com a estratificação, ainda sim não foi completamente resolvido.

Abaixo, completamos a análise com o fornecimento do relatório das médias dos diferentes classificadores durante o treino e validação:

```
----------------------------------------------------------------------
Acurácia do conjunto de treinamento:
----------------------------------------------------------------------
Média de acurácia para kernel POLY
Treino:  0.9480154486036839
Validação:  0.7147058823529412

Média de acurácia para kernel POLY (sem atributos correlacionados)
Treino:  0.7986953569306509
Validação:  0.6183660130718954

Média de acurácia para kernel RBF
Treino:  0.8812197606315253
Validação:  0.7250980392156862

Média de acurácia para kernel RBF (sem atributos correlacionados)
Treino:  0.7745216874628639
Validação:  0.6407843137254902

Média de acurácia para kernel SIGMOID
Treino:  0.7352762923351159
Validação:  0.6431372549019607

Média de acurácia para kernel SIGMOID (sem atributos correlacionados)
Treino:  0.6032569391392921
Validação:  0.5481699346405229

Média de acurácia para kernel LINEAR
Treino:  0.7456777862660217
Validação:  0.6525490196078431

Média de acurácia para kernel LINEAR (sem atributos correlacionados)
Treino:  0.615208386384857
Validação:  0.5621568627450981
```

Relatório 2. Médias das acurácias dos diferentes kernels durante o treino e validação K-fold.

REFERÊNCIAS:

https://scikit-learn.org/stable/modules/svm.html

https://www.youtube.com/watch?v=Zj1CoJk2feE

https://scikit-learn.org/stable/modules/multiclass.html

https://archive.ics.uci.edu/ml/datasets/glass+identification

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

https://www.vebuso.com/2020/03/svm-hyperparameter-tuning-using-gridsearchcv/

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RepeatedStratifiedKFold.html