Relatório de progresso no projeto final – Entrega 5

Andrey Moutelik, arma2

Recapitulando...

Estou trabalhando com o dataset Swarm Behaviour, que na realidade contém 3 datasets, cada um determinando se uma simulação de 'boids' está alinhada, reunida ou agrupada. Em teoria, os dados (com exceção da classe final) deveriam ser iguais, porém enquanto isso é o caso para 'Aligned' e 'Grouped', os dados de 'Flocking' são parcialmente diferentes. Esta diferença é suficiente para desconsiderar o dataset em comparações com os outros dois, a quantidade de entradas distintas é muito grande.

Para fins de simplicidade, optei por trabalhar apenas com o dataset 'Aligned'. O formato dos dados é idêntico aos outros dois, portanto todos os scripts funcionam para os 3 datasets. Cada linha do dataset contém 2400 features, e classifica uma simulação de acordo com o critério determinado. No caso de 'Aligned', a classe pode ser '0': Not Aligned ou '1': Aligned.

Faço o balanceamento dos dados utilizando uma mistura de Undersampling e Oversampling, e então aplico PCA para reduzir a dimensionalidade do dataset. Devido a recomendações vistas em artigos, e a fim de reduzir overfitting (um problema que ao meu ver tem sido constante nos passos a seguir) opto uma redução de cerca de 90% na dimensionalidade para 250 features, mantendo cerca de 82% da variância.

Por fim tenho uma função que recebe um modelo e os conjuntos de treino e teste, faz a validação cruzada de acordo com as especificações do projeto, imprime a acurácia, o classification report e o desvio padrão, além de retornar os dados relevantes para um dicionário para uso futuro.

Variação paramétrica dos modelos requisitados na especificação do projeto:

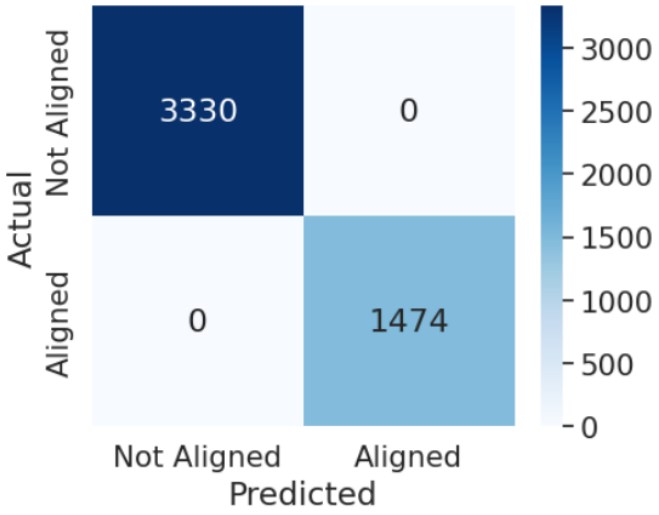
K-NN

Eu tinha preparado um algoritmo para escolher o melhor 'n' para o dataset, mas depois de rodar valores de ´1' a '31' ele retornou que o melhor valor de K é 1, e quando rodo o algoritmo tenho como resultado:

```
from sklearn.neighbors import KNeighborsClassifier

# Create a KNeighborsClassifier object
knn = KNeighborsClassifier(n_neighbors=1)

knn_dict = run_model(knn, X_train, y_train, X_test, y_test)
```

```
Train scores: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Mean train score: 1.000

Test scores: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Mean test score: 1.000
Standard deviation of test scores: 0.000

Accuracy: 1.000
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3330
           1       1.00      1.00      1.00      1474

    accuracy                           1.00      4804
   macro avg       1.00      1.00      1.00      4804
weighted avg       1.00      1.00      1.00      4804
```

Com a seguinte confusion matrix:



Fiquei honestamente perplexo, pelos meus conhecimentos o K do K-NN é o principal hiperparâmetro e mesmo com 1 o classificador é um 'fit perfeito'. O processo de seleção e ajuste de parâmetros, da forma como é descrito no artigo https://www.kdnuggets.com/2020/05/hyperparameter-optimization-machine-learning-models.html não faz sentido nesse caso, pois seria retornado apenas o K com o valor de 1 e os parâmetros em seu estado default.

O problema não está na função que fiz para rodar os modelos, pois uma implementação tradicional do knn também resulta nesta acurácia:

```python
from sklearn.neighbors import KNeighborsClassifier

# Create a KNeighborsClassifier object
knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(X_train, y_train)

# Make predictions on test data
y_pred = knn.predict(X_test)

# Calculate accuracy score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

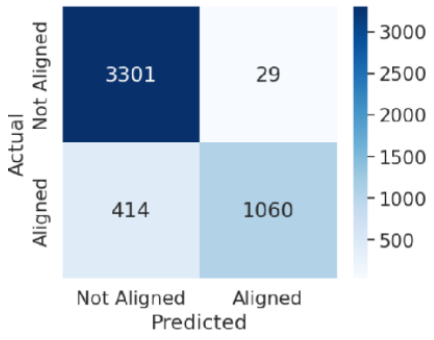
Eu não sei para onde ir com esse modelo em específico. Posso ajustar os parâmetros a fim de reduzir um suposto 'overfitting' (o que inclusive faço em modelos seguintes), mas com relação a escolha de parâmetros por grid search, random search ou outro método, neste caso não posso fazer nada.

LVQ

Após instalar o pacote de LVQ do sk_learn no colab (ele não está podendo ser importado da forma tradicional).

Uma primeira execução do LVQ resultou no seguinte classification report e matriz de confusão:

```
Train scores: [0.857, 0.834, 0.851, 0.851, 0.67, 0.851, 0.852, 0.831, 0.671, 0.826]
Mean train score: 0.809

Validation scores: [0.842, 0.835, 0.867, 0.843, 0.671, 0.844, 0.86, 0.82, 0.681, 0.835]
Mean validation score: 0.810

Test scores: [0.922, 0.914, 0.918, 0.915, 0.813, 0.915, 0.919, 0.91, 0.813, 0.908]
Mean test score: 0.895
Standard deviation of test scores: 0.041

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.99      0.94      3330
           1       0.97      0.72      0.83      1474

    accuracy                           0.91      4804
   macro avg       0.93      0.86      0.88      4804
weighted avg       0.91      0.91      0.90      4804
```



A fim de ajustar os hiperparâmetros, realizei o seguinte GridSearch:

```
[36] from sklearn_lvq import GlvqModel
     from sklearn.model_selection import GridSearchCV

     # Define the hyperparameter grid
     param_grid = {
         'prototypes_per_class': [2, 3, 4],
         'beta': [1, 2, 3],
         'max_iter': [1000, 2500, 5000],
     }

     # Initialize the LVQ model
     lvq = GlvqModel()

     # Create the grid search object
     grid_search = GridSearchCV(lvq, param_grid=param_grid, cv=3, n_jobs=-1, verbose=1)

     # Fit the grid search to the data
     grid_search.fit(X_train, y_train)

     # Print the best hyperparameters and accuracy score
     print("Best parameters:", grid_search.best_params_)
     print("Best score:", grid_search.best_score_)
```

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits
Best parameters: {'beta': 3, 'max_iter': 2500, 'prototypes_per_class': 4}
Best score: 0.8560658338899029
```

Levando em conta que tenho os seguintes parâmetros disponíveis para ajuste:

prototypes_per_class: This parameter controls the number of prototypes used for each class.

max_iter: This parameter determines the maximum number of iterations for the optimization algorithm to converge.

gtol: This parameter determines the tolerance for the change in the objective function value that indicates convergence. Common values are between 1e-6 and 1e-3.

beta: This parameter controls the learning rate of the optimization algorithm. Common values are between 1 and 10.

c: This parameter controls the regularization strength.

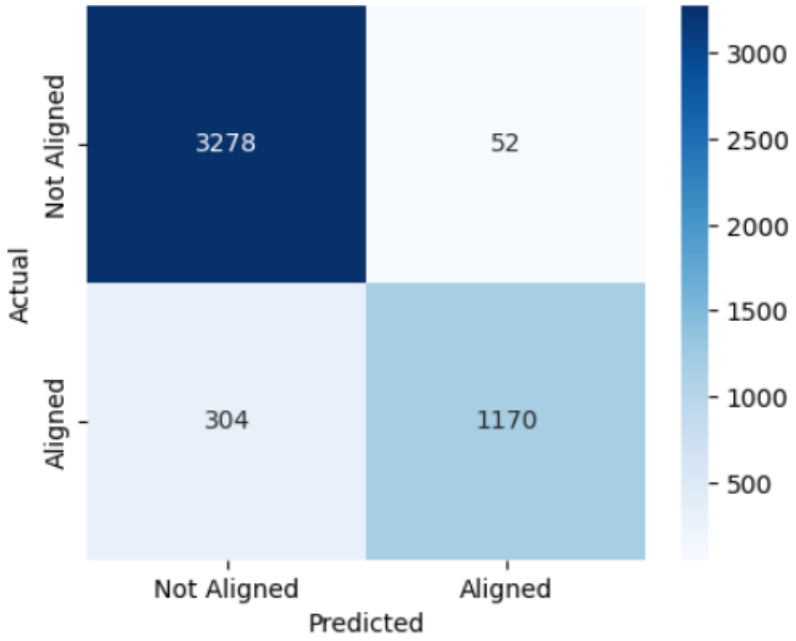
random_state: This parameter controls the random seed used for the initialization of the algorithm.

Nesta execução eu usei apenas 3 como o valor para cross validation, a fim de reduzir um pouco o longo tempo da busca. Os valores e parâmetros foram escolhidos para a grid search com base nos valores comuns associados a eles.

De qualquer forma, usando os valores descobertos no grid search, executo o modelo novamente:

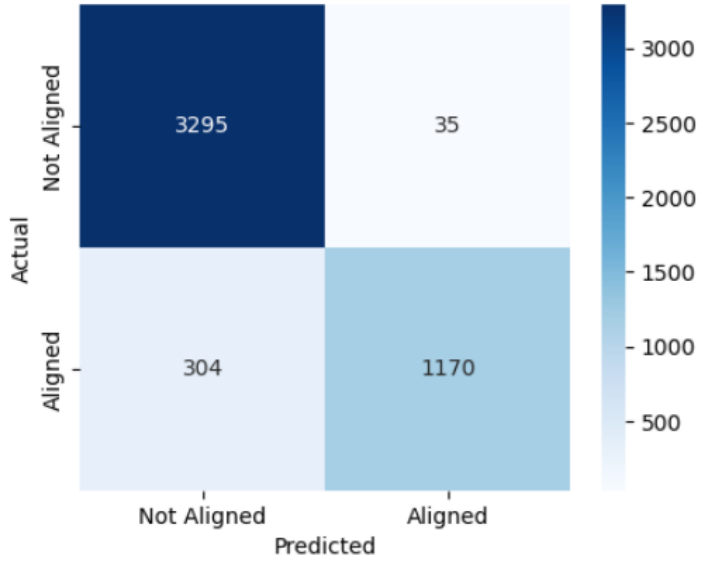
```
from sklearn_lvq import GlvqModel

lvq = GlvqModel(prototypes_per_class=4, initial_prototypes=None, max_iter=2500, gtol=1e-05, beta=3, c=None, random_state=42)

lvq_dict = run_model(lvq, X_train, y_train, X_test, y_test)
```

```
Train scores: [0.906, 0.852, 0.852, 0.906, 0.902, 0.9, 0.851, 0.85, 0.853, 0.851]
Mean train score: 0.872

Test scores: [0.891, 0.849, 0.86, 0.9, 0.894, 0.897, 0.858, 0.843, 0.862, 0.865]
Mean test score: 0.872
Standard deviation of test scores: 0.020

Accuracy: 0.926
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.98      0.95      3330
           1       0.96      0.79      0.87      1474

    accuracy                           0.93      4804
   macro avg       0.94      0.89      0.91      4804
weighted avg       0.93      0.93      0.92      4804
```



Temos notavelmente uma melhora na acurácia, o que é refletido na matriz de confusão.

Em seguida ainda alterei o parâmetro gtol para 1e-03, o que resultou em uma redução de acurácia para 88%, e em seguida testei como 1e-06. O modelo que levava cerca de 3 minutos para rodar levou 15 devido a isso, mas houve uma pequena melhora na acurácia:



Árvore de decisão

Para este modelo, como de praxe, realizo uma execução sem determinar os parâmetros:

```
[44] from sklearn.tree import DecisionTreeClassifier

     dt = DecisionTreeClassifier(random_state=42)

     dt_dict = run_model(dt, X_train, y_train, X_test, y_test)
```

```
Train scores: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Mean train score: 1.000

Test scores: [0.999, 0.999, 0.999, 0.999, 0.999, 1.0, 0.999, 0.999, 0.999, 0.999]
Mean test score: 0.999
Standard deviation of test scores: 0.000

Accuracy: 0.949
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.93      0.96      3330
           1       0.87      0.99      0.92      1474

    accuracy                           0.95      4804
   macro avg       0.93      0.96      0.94      4804
weighted avg       0.95      0.95      0.95      4804
```

```
[47] print("Maximum depth of decision tree:", dt.tree_.max_depth)
```

```
Maximum depth of decision tree: 11
```

Temos uma acurácia de 94,9% e uma profundidade de 11.

Em seguida executo um Grid Search com valores e parâmetros comumente utilizados em árvores de decisão, e tenho o seguinte resultado:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Create a decision tree classifier object
dt = DecisionTreeClassifier(random_state=42)

# Define the grid search parameters
param_grid = {'max_depth': [5, 10, 20, 50, None],
              'min_samples_split': [2, 5, 10, 20],
              'min_samples_leaf': [1, 2, 4, 8],
              'max_features': ['sqrt', 'log2', None]}

# Perform the grid search
grid_search = GridSearchCV(dt, param_grid, cv=5, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)

# Print the best parameters and score
print("Best parameters found: ", grid_search.best_params_)
print("Best score: ", grid_search.best_score_)
```

```
Fitting 5 folds for each of 240 candidates, totalling 1200 fits
Best parameters found:  {'max_depth': 20, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best score:  0.9992455020313408
```

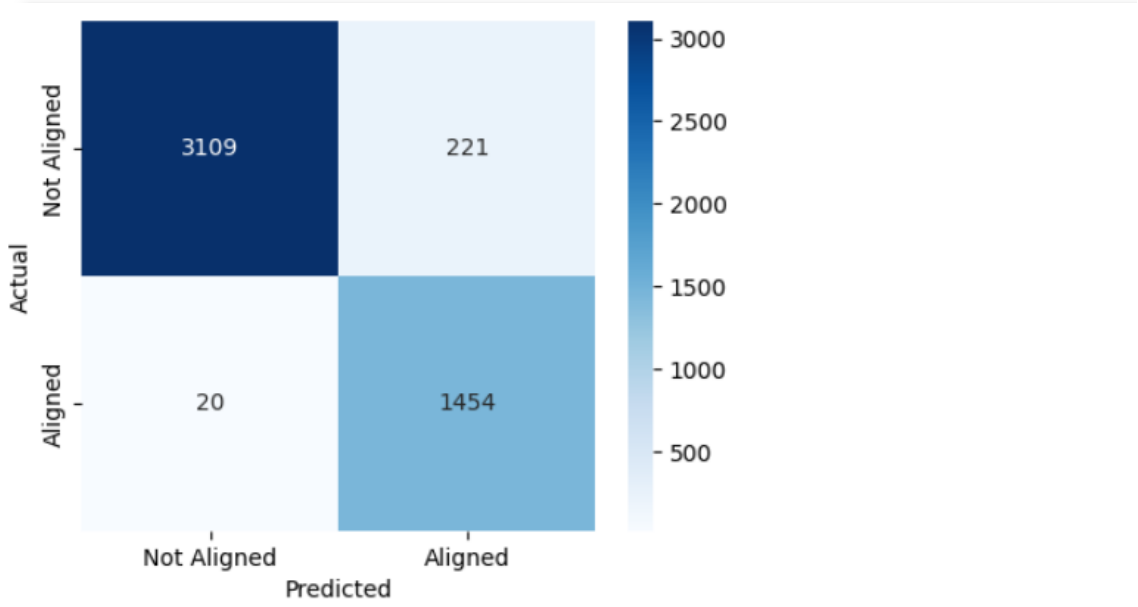
Executando o modelo com os parâmetros encontrados, temos como resultado:

```
[51] from sklearn.tree import DecisionTreeClassifier

     dt = DecisionTreeClassifier(max_depth=20, min_samples_leaf=1, min_samples_split = 2, max_features = None, random_state=42)

     dt_dict = run_model(dt, X_train, y_train, X_test, y_test)
```

```
Train scores: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Mean train score: 1.000

Test scores: [0.999, 0.999, 0.999, 0.999, 0.999, 1.0, 0.999, 0.999, 0.999, 0.999]
Mean test score: 0.999
Standard deviation of test scores: 0.000

Accuracy: 0.949
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.93      0.96      3330
           1       0.87      0.99      0.92      1474

    accuracy                           0.95      4804
   macro avg       0.93      0.96      0.94      4804
weighted avg       0.95      0.95      0.95      4804
```

```
[52] print("Maximum depth of decision tree:", dt.tree_.max_depth)
```

```
Maximum depth of decision tree: 11
```



Considerando que os valores encontrados eram quase todos o padrão de inicialização de Decision Trees, o resultado que temos é exatamente igual ao que tínhamos antes, quando iniciamos a árvore sem parâmetro algum. O que pode ser feito agora é ajustar os parâmetros a fim de tornar o encaixe da árvore mais geral. Uma execução da árvore onde reduzo a profundidade máxima para 5, e dobro os valores de min_samples_leaf e min_samples_split termina com um resultado quase igual:

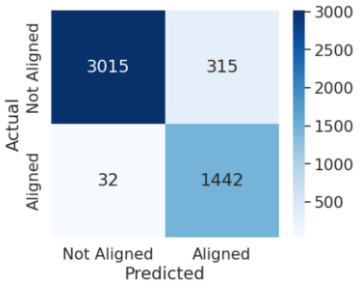
```
[71] from sklearn.tree import DecisionTreeClassifier

     dt = DecisionTreeClassifier(max_depth=5, min_samples_leaf=2, min_samples_split = 4, max_features = None, random_state=42)

     dt_dict = run_model(dt, X_train, y_train, X_test, y_test)

     Train scores: [0.995, 0.995, 0.995, 0.996, 0.995, 0.996, 0.995, 0.994, 0.996, 0.995]
     Mean train score: 0.995

     Test scores: [0.994, 0.997, 0.994, 0.992, 0.994, 0.994, 0.992, 0.992, 0.995, 0.995]
     Mean test score: 0.994
     Standard deviation of test scores: 0.002

     Accuracy: 0.949
     Classification Report:
                   precision    recall  f1-score   support

                0       0.99      0.93      0.96      3330
                1       0.87      0.99      0.92      1474

         accuracy                           0.95      4804
        macro avg       0.93      0.96      0.94      4804
     weighted avg       0.95      0.95      0.95      4804
```

Nesse caso, acredito que para compensar a profundidade reduzida a árvore ficou mais larga. O atributo que mais importa portanto nessa execução, e que suponho que reduz ao máximo overfitting é o 'max_features', que reduz bastante a acurácia do modelo se trocado para 'sqrt' ou 'log2', um grid_search ou random_search com um número limitado de atributos pode se provar interessante na entrega seguinte. Apreciaria feedback com relação a isso.

SVM

Com relação a SVM, confesso não ter realizado uma busca por hiperparâmetros como orientado. Primeiramente testei os diferentes kernels com o resto dos parâmetros default:

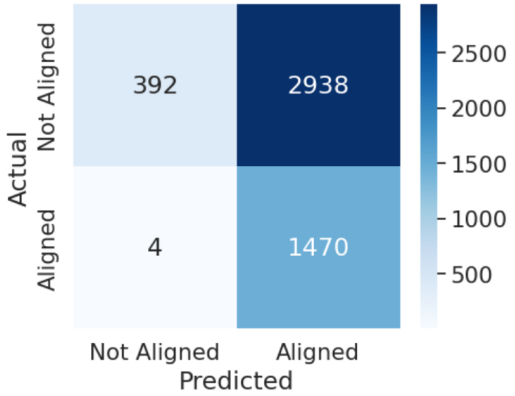
Linear, que apresentou um resultado razoavél:



Poly, que depois de **23 minutos** de execução apresentou um resultado desastroso:

```
[50] from sklearn.svm import SVC

     svm = SVC(kernel='poly')

     svm_dict = run_model(svm, X_train, y_train, X_test, y_test)
```

```
Train scores: [0.557, 0.559, 0.977, 0.559, 0.561, 0.562, 0.56, 0.559, 0.558, 0.558]
Mean train score: 0.601

Test scores: [0.553, 0.565, 0.981, 0.57, 0.561, 0.564, 0.554, 0.558, 0.555, 0.548]
Mean test score: 0.601
Standard deviation of test scores: 0.127

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.12      0.21      3330
           1       0.33      1.00      0.50      1474

    accuracy                           0.39      4804
   macro avg       0.66      0.56      0.36      4804
weighted avg       0.79      0.39      0.30      4804
```



E então RBF, que foi um ‘fit perfeito’:

```
from sklearn.svm import SVC

svm = SVC(kernel='rbf', random_state=42)

svm_dict = run_model(svm, X_train, y_train, X_test, y_test)
```

```
Train scores: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Mean train score: 1.000

Test scores: [1.0, 0.999, 1.0, 1.0, 0.999, 1.0, 1.0, 1.0, 0.999, 0.999]
Mean test score: 1.000
Standard deviation of test scores: 0.000

Accuracy: 1.000
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3330
           1       1.00      1.00      1.00      1474

    accuracy                           1.00      4804
   macro avg       1.00      1.00      1.00      4804
weighted avg       1.00      1.00      1.00      4804
```

Eu tinha lido em um artigo sobre SVM que o kernel rbf era propenso a overfitting. Mas devido a esse resultado, optei por explorar mais esse kernel, alterando os valores dos parâmetros ‘C’ e ‘gamma’ manualmente a fim de generalizar o modelo. Novamente, uma busca pelos melhores parâmetros nesse caso não faz muito sentido pois retornaria os valores padrão com o kernel rbf.

C = 0.1 e gamma = 0.01:

```
[57] from sklearn.svm import SVC

     svm = SVC(kernel='rbf', C= 0.1, gamma= 0.01, random_state=42)

     svm_dict = run_model(svm, X_train, y_train, X_test, y_test)
```

```
Train scores: [0.998, 0.998, 0.999, 0.998, 0.998, 0.998, 0.998, 0.998, 0.999, 0.998]
Mean train score: 0.998

Test scores: [0.998, 0.999, 0.998, 0.998, 0.998, 0.999, 0.999, 0.999, 0.997, 0.999]
Mean test score: 0.998
Standard deviation of test scores: 0.001

Accuracy: 1.000
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3330
           1       1.00      1.00      1.00      1474

    accuracy                           1.00      4804
   macro avg       1.00      1.00      1.00      4804
weighted avg       1.00      1.00      1.00      4804
```

Redução do gamma para 0.001

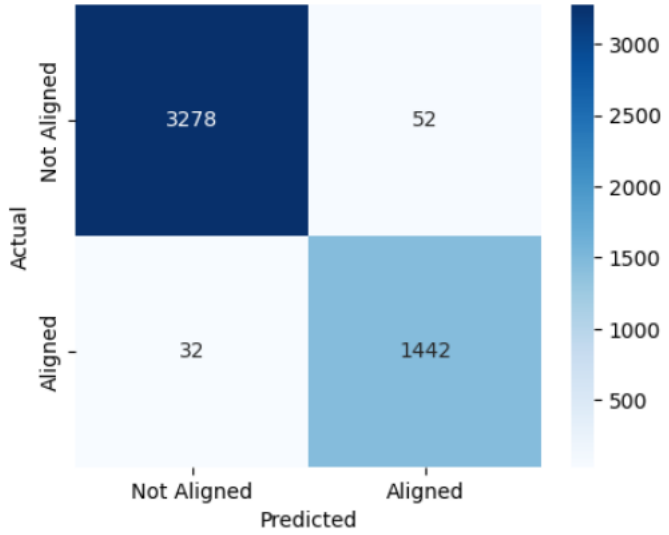
```
[ ]  from sklearn.svm import SVC

     svm = SVC(kernel='rbf', C= 0.1, gamma= 0.001, random_state=42)

     svm_dict = run_model(svm, X_train, y_train, X_test, y_test)
```

```
Train scores: [0.873, 0.873, 0.873, 0.873, 0.871, 0.874, 0.872, 0.874, 0.874, 0.872]
Mean train score: 0.873

Test scores: [0.86, 0.873, 0.88, 0.864, 0.865, 0.868, 0.878, 0.873, 0.875, 0.885]
Mean test score: 0.872
Standard deviation of test scores: 0.007

Accuracy: 0.881
Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.91      0.91      3330
           1       0.80      0.82      0.81      1474

    accuracy                           0.88      4804
   macro avg       0.86      0.86      0.86      4804
weighted avg       0.88      0.88      0.88      4804
```

Temos uma redução razoável na acurácia devido a essa mudança. Com a intenção de encontrar um balanço entre performance e generalização optei por aumentar o valor do gamma para 0.005:

```
from sklearn.svm import SVC

svm = SVC(kernel='rbf', C= 0.1, gamma= 0.005, random_state=42)

svm_dict = run_model(svm, X_train, y_train, X_test, y_test)
```

```
Train scores: [0.983, 0.983, 0.983, 0.983, 0.983, 0.983, 0.983, 0.983, 0.983, 0.983]
Mean train score: 0.983

Test scores: [0.982, 0.983, 0.985, 0.981, 0.981, 0.979, 0.989, 0.981, 0.981, 0.985]
Mean test score: 0.983
Standard deviation of test scores: 0.003

Accuracy: 0.983
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.98      0.99      3330
           1       0.97      0.98      0.97      1474

    accuracy                           0.98      4804
   macro avg       0.98      0.98      0.98      4804
weighted avg       0.98      0.98      0.98      4804
```



A busca por parâmetros não foi como especificado, mas considerando o tempo de execução do modelo (chegando a 15~20 minutos) que inviabiliza uma busca por grade, além da maldição desse dataset que gosta de retornar acurácia de 100% em determinados modelos, fiquei satisfeito com os parâmetros encontrados. Espero que tenha sido aceitável.

Random Forest

Para Random Forest, tentei fazer um grid search inicialmente com diferentes valores para n_estimators, max_depth, min_samples_split e min_samples_leaf, mas logo percebi que provavelmente ia levar tempo demais. Uma execução do modelo sem hiperparâmetros resulta em:

```
[79] from sklearn.ensemble import RandomForestClassifier

     rf = RandomForestClassifier()

     rf_dict = run_model(rf, X_train, y_train, X_test, y_test)
```

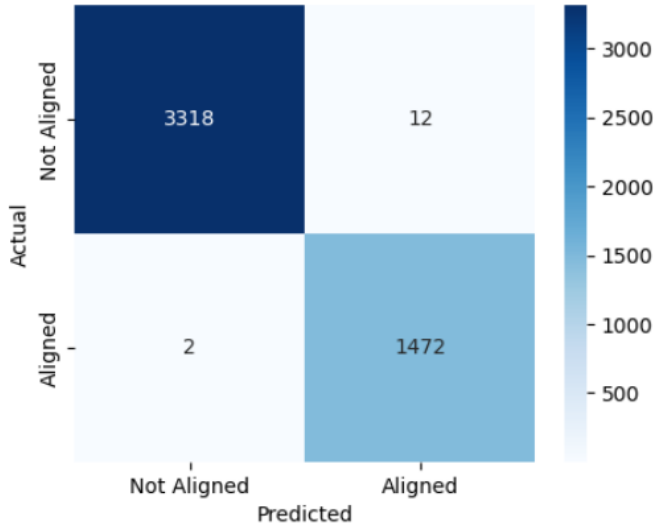
```
Train scores: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Mean train score: 1.000

Test scores: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Mean test score: 1.000
Standard deviation of test scores: 0.000

Accuracy: 1.000
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3330
           1       1.00      1.00      1.00      1474

    accuracy                           1.00      4804
   macro avg       1.00      1.00      1.00      4804
weighted avg       1.00      1.00      1.00      4804
```

E uma execução com parâmetros designados a reduzir overfitting resultam em algo semelhante:

```
[77] from sklearn.ensemble import RandomForestClassifier

     rf = RandomForestClassifier(n_estimators=100, random_state=42, max_depth=5, min_samples_split=4, min_samples_leaf=2, max_features = 'sqrt')

     rf_dict = run_model(rf, X_train, y_train, X_test, y_test)
```

```
Train scores: [0.998, 0.998, 0.999, 0.998, 0.998, 0.998, 0.999, 0.998, 0.998, 0.999]
Mean train score: 0.998

Test scores: [0.998, 0.998, 0.999, 0.997, 0.999, 0.996, 0.999, 0.998, 0.996, 0.998]
Mean test score: 0.998
Standard deviation of test scores: 0.001

Accuracy: 0.997
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3330
           1       0.99      1.00      1.00      1474

    accuracy                           1.00      4804
   macro avg       1.00      1.00      1.00      4804
weighted avg       1.00      1.00      1.00      4804
```

Acredito que a pequena perda de acurácia compensa o tempo de execução menor e a maior generalização do modelo



Por fim, para essa entrega, fiz

MLP

Rodando sem determinação de parâmetros tenho como resultado:

```
[60] from sklearn.neural_network import MLPClassifier

     mlp = MLPClassifier(random_state=42)

     mlp_dict = run_model(mlp, X_train, y_train, X_test, y_test)
```

```
Train scores: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Mean train score: 1.000

Test scores: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Mean test score: 1.000
Standard deviation of test scores: 0.000

Accuracy: 1.000
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3330
           1       1.00      1.00      1.00      1474

    accuracy                           1.00      4804
   macro avg       1.00      1.00      1.00      4804
weighted avg       1.00      1.00      1.00      4804
```

Nunca pensei que ia ficar tão frustrado com esses classificadores “perfeitos”. Otimista com o tempo de execução do mlp (1 minuto), resolvi executar Random Search por parâmetros para ver no que dava.

DUAS HORAS DEPOIS, tive o seguinte resultado:

```
[61] from sklearn.neural_network import MLPClassifier
     from sklearn.model_selection import RandomizedSearchCV
     from scipy.stats import randint as sp_randint

     # define the parameter space
     param_dist = {'hidden_layer_sizes': sp_randint(50, 500),
                   'activation': ['relu', 'tanh'],
                   'solver': ['adam', 'lbfgs', 'sgd'],
                   'alpha': [0.0001, 0.001, 0.01],
                   'learning_rate': ['constant', 'adaptive'],
                   'max_iter': sp_randint(500, 2000)}

     # create the MLP classifier object
     mlp = MLPClassifier()

     # create the randomized search object
     clf = RandomizedSearchCV(mlp, param_distributions=param_dist,
                              n_iter=50, cv=5, scoring='accuracy',
                              n_jobs=-1, random_state=42, verbose=2)

     # fit the randomized search object to the data
     clf.fit(X_train, y_train)

     # print the best parameters and score
     print("Best parameters: ", clf.best_params_)
     print("Best score: ", clf.best_score_)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
Best parameters:  {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': 320, 'learning_rate': 'constant', 'max_iter': 1595, 'solver': 'adam'}
Best score:  1.0
```


```
Best parameters:  {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': 320, 'learning_rate': 'constant', 'max_iter': 1595, 'solver': 'adam'}
Best score:  1.0
```


Sem surpresa nenhuma, o a execução do MLP com esses parâmetros resultou em 100% de acurácia:

```
[63] from sklearn.neural_network import MLPClassifier

     mlp = MLPClassifier(activation= 'relu', alpha= 000.1, hidden_layer_sizes= 320, learning_rate='constant', max_iter=1500, solver = 'adam', random_state=42)

     mlp_dict = run_model(mlp, X_train, y_train, X_test, y_test)
```

```
Train scores: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Mean train score: 1.000

Test scores: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Mean test score: 1.000
Standard deviation of test scores: 0.000

Accuracy: 1.000
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3330
           1       1.00      1.00      1.00      1474

    accuracy                           1.00      4804
   macro avg       1.00      1.00      1.00      4804
weighted avg       1.00      1.00      1.00      4804
```

Eu acredito que nesse caso, eu devo fazer como fiz com SVM e tentar generalizar o modelo a fim de combater overfitting. Apreciaria feedback para ver se devo seguir este caminho.

Ainda não realizei a execução de Comitê de Redes Neurais Artificiais e Comitê Heterogêneo. De acordo com a agenda da disciplina ainda temos uma entrega de variação paramétrica no dia 03/05, então estou optando por deixar esses dois modelos, além de mais refinamentos dos parâmetros dos modelos aqui apresentados, para essa próxima entrega. Apreciaria saber se estou indo pelo caminho certo, e honestamente o que interpretar desses resultados de acurácia de 100%.

Como uma consideração final, vendo a descrição do projeto final novamente, uma das etapas é “Extrair uma árvore de decisão e identificar quais são os primeiros atributos utilizados na construção da árvore.

❖ Propor sugestões de decisões com base nos atributos encontrados”

No meu caso acho essa etapa impossível. Já começo com 2400 features, então fazer decisões sobre a influência de uma sobre o resultado final é complicado. Em seguida eu aplico PCA, com as features resultantes sendo uma incógnita para todos nós.

-Andrey Moutelik