Entrega 4 - Variação paramétrica - Relatório sobre evoluções da variação paramétrica

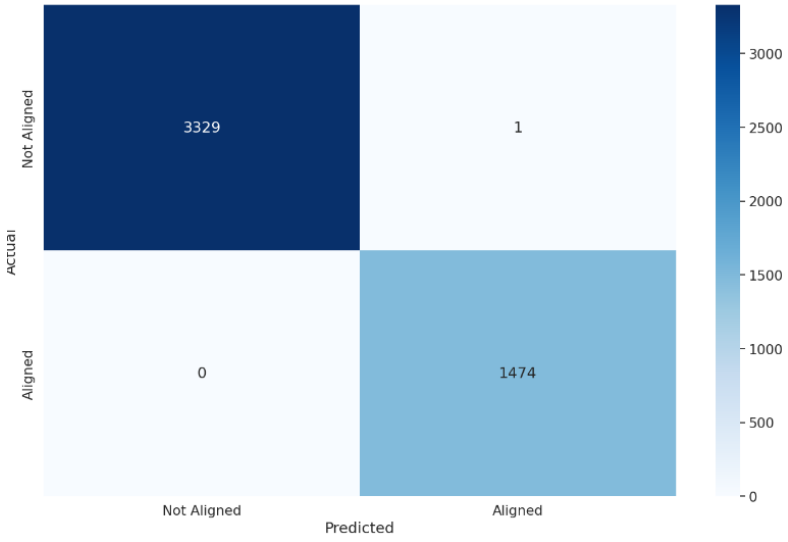
Andrey Moutelik (arma2)

Nesta etapa do projeto, a fim de estabelecer um parâmetro base, (o que deveria ser feito antes da fase de processamento, mas terminei não o fazendo). Utilizei os dados crus do dataset em dois modelos: Regressão Logística e SVM.

Como resultado temos um “encaixe perfeito”:

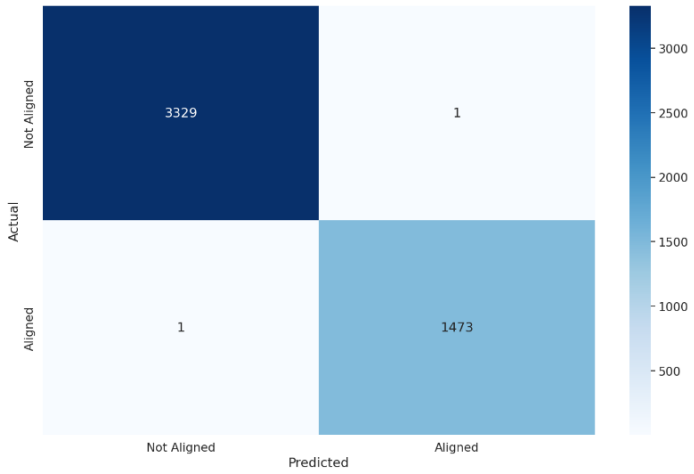
Regressão Logística

```
Training scores for each fold: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Testing scores for each fold: [1.0, 0.9994797086368367, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Mean testing score:, 1.0
Standard deviation of testing scores:, 0.000156
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

Accuracy score:, 1.0
```



SVM:

```
Training scores for each fold: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Testing scores for each fold: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Mean testing score:, 1.0
Standard deviation of testing scores:, 0.0
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

Accuracy score:, 1.0
```



É evidente que num caso como esse está havendo overfitting. Eu gostaria de ter como provar isso mas mesmo depois de horas de pesquisa ainda não sei exatamente como. Assumo que seja mais devido aos indicadores (número absurdo de features, longo tempo de processamento..)

Enfim, a existência de overfitting é algo que eu devo ter como verdade se não tudo o que eu fizer daqui pra frente estaria apenas “piorando” o modelo.

Como adendo a esse quesito, eu gostaria de deixar explicito que testei esses modelos de diversas maneiras, com e sem cross validation, diferentes implementações.. Sempre com o mesmo resultado. Não acredito que haja data leakage, e também avaliei a correlação das features com as classes e não há problemas nesse aspecto.

Em seguida, como decidi que meu foco principal é reduzir o overfitting nos modelos, optei por reduzir o número de features do PCA de 300 pra 250, mantendo ~82% da variância original mas reduzindo a dimensionalidade para ~10% da original.

Para a aplicação dos modelos eu optei, depois de algumas horas de trabalho, por definir uma função que recebe um modelo e os subsets de treino e teste, e retorna um dicionário com os dados importantes.

```
def run_model(model, X_train, y_train, X_test, y_test, n_folds=10):
    # create empty lists to store scores and predictions for each fold
    train_scores = []
    test_scores = []
    y_preds = []

    # create stratified k-fold cross-validation object
    skf = StratifiedKFold(n_splits=n_folds, shuffle=True, random_state=42)

    # loop over each fold
    for train_index, test_index in skf.split(X_train, y_train):
        # split data into training and validation sets for this fold
        X_train_fold, y_train_fold = X_train[train_index], y_train[train_index]
        X_val_fold, y_val_fold = X_train[test_index], y_train[test_index]

        # fit model on training set for this fold
        model.fit(X_train_fold, y_train_fold)

        # calculate and store training and validation scores for this fold
        train_score = model.score(X_train_fold, y_train_fold)
        test_score = model.score(X_val_fold, y_val_fold)
        train_scores.append(train_score)
        test_scores.append(test_score)

        # predict target variable for test set for this fold and store predictions
        y_pred = model.predict(X_test)
        y_preds.append(y_pred)
```

```
# calculate mean training and validation scores over all folds
train_scores_mean = np.mean(train_scores)
test_scores_mean = np.mean(test_scores)

# create dictionary of results
results = {
    "train_scores": train_scores,
    "test_scores": test_scores,
    "y_preds": y_preds,
    "train_scores_mean": train_scores_mean,
    "test_scores_mean": test_scores_mean,
}

# print out scores and classification report
print("Train scores:", [round(score, 3) for score in train_scores])
print(f"Mean train score: {train_scores_mean:.3f}\n")
print("Test scores:", [round(score, 3) for score in test_scores])
print(f"Mean test score: {test_scores_mean:.3f}")
print(f"Standard deviation of test scores: {np.std(test_scores):.3f}\n")
y_test_pred = results["y_preds"][-1]
print("Classification Report:")
print(classification_report(y_test, y_test_pred))

return results
```

Ainda pretendo melhorar essa função se houver necessidade (provavelmente precisarei adicionar mais métricas no dicionário de resultados, por exemplo), mas acredito que a implementação do cross validation está de acordo com o que foi requisitado nas especificações do projeto.

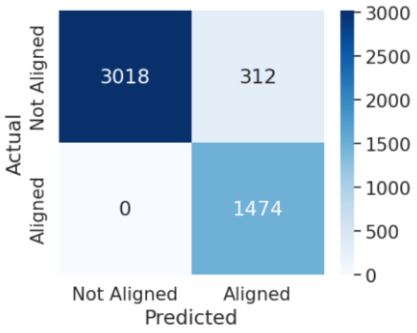
Também fiz uma função que produz a confusion matrix com base no dicionário, além de receber um parâmetro que indica qual dataset está sendo usado:

```
def plot_confusion_matrix(results, y_test, datasetNum):
    y_pred = results["y_preds"][-1]
    cm = confusion_matrix(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(5, 4))
    if(datasetNum == 1):
       sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', xticklabels=['Not Aligned', 'Aligned'], yticklabels=['Not Aligned', 'Aligned'])
    elif (datasetNum == 2):
       sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', xticklabels=['Not Flocking', 'Flocking'], yticklabels=['Not Flocking', 'Flocking'])
    else:
       sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', xticklabels=['Not Grouped', 'Grouped'], yticklabels=['Not Grouped', 'Grouped'])
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```

Com base nessas funções, testei os seguintes modelos:

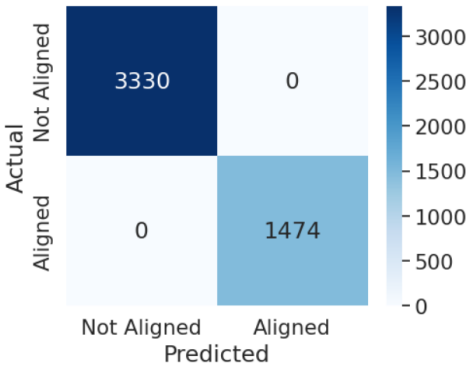
Regressão Logística:

```
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, 1.0, 0.999, 1.0, 1.0, 0.999, 1.0, 1.0, 0.997, 0.999]
Mean test score: 0.999
Standard deviation of test scores: 0.001

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.91      0.95      3330
           1       0.83      1.00      0.90      1474

    accuracy                           0.94      4804
   macro avg       0.91      0.95      0.93      4804
weighted avg       0.95      0.94      0.94      4804
```



K-NN

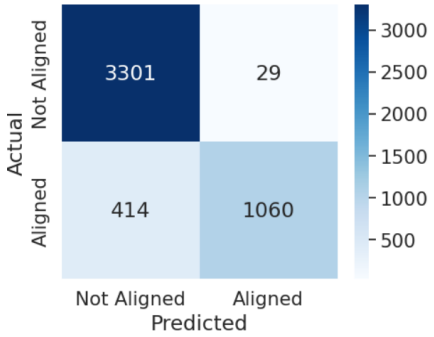
```
Train scores: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Mean train score: 1.000

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

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
```



Vale notar que eu tinha elaborado um script para definir o melhor k para o algoritmo, mas o score terminou sendo perfeito independente do k escolhido (terminei usando k = 1 mesmo), os subsets usados passaram pelo balanceamento e pca, então não há muito mais que eu possa fazer. Pode haver um erro na forma como eu avalio os modelos, não sei. Por hora, aparentemente k-nn é um fit perfeito.

LVQ

```
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
```



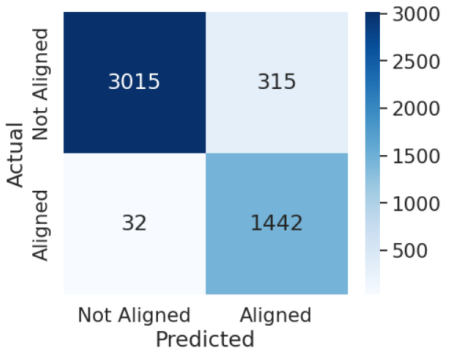
(tive que instalar a biblioteca do LVQ por fora)

Finalmente um resultado promissor, no sentido de haver o que melhorar ajustando hiperparâmetros. Não fiz esses ajustes ainda, mas acredito que a performance do modelo possa ser levemente melhor.

Vale notar que esse foi o primeiro modelo onde os scores do cross validation não foram em geral [1.0] ou [0.99], o que me animou um pouco

Por fim, SVM

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

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.91      0.95      3330
           1       0.82      0.98      0.89      1474

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



Devido às circunstâncias peculiares deste dataset, fiquei atrasado com relação a edição dos hiperparâmetros dos modelos, o que pretendo compensar na entrega seguinte.

Por outro lado, a criação dessa função modular para a execução de modelos agilizará bastante o processo de teste de diferentes parâmetros de agora em diante. Além disso deixa o código menos bagunçado e extenso.