Instituto Superior de Engenharia de Coimbra
Engenharia Informática
Inteligência Computacional

notMNIST Data Set

Cuckoo Search Algorithm

Inteligência Computacional

2021/2022

Projeto - Fase III

Diogo Dias Lopes– a2018019746
Leonardo Marques - a2019123778

Conteúdo

Introdução

Este projeto tem como objetivo a implementação e validação de uma das técnicas de inteligência computacional mais concretamente um algoritmo de inteligência *Swarm*, estudado anteriormente na Fase II – Seminário deste Projeto, a um caso de estudo real, já analisado na Fase I.

A técnica de inteligência Swarm utilizada e implementada junto com o caso de estudo foi o Cuckoo Search Algorithm, desenvolvida por Xin-She Yang e Suash Deb em 2009, um algoritmo meta heurístico para otimização global baseado no comportamento de parasitagem dos cucos aquando da reprodução da sua espécie.

O caso de estudo analisado na Fase I foi o dataset notMNIST, sendo um tema com utilidade, com muito interesse e que nos permite a aplicação desta técnica.

Após a analise do caso de estudo e do estudo do algoritmo CSO segue-se a fase de otimização da arquitetura com base nas metodologias estudadas. Foram selecionados dois Hiper parâmetros sendo que utilizamos um modelo de otimização baseado num dos classificadores estudados na disciplina, o Random Forest Classifier aliado ao algoritmo anteriormente referido, o Cuckoo Search Algorithm e comparamos este mesmo modelo com um modelo *benchmark*, o GridSearchCV e retiramos as conclusões finais desta mesma comparação.

O presente documento está organizado da seguinte forma:

- Capítulo I: Descrição do problema;
- Capítulo II: Descrição de todas as metodologias utilizadas;
- Capítulo III: Diagrama de classes;
- Capítulo IV: Descrição da implementação dos algoritmos;
- Capítulo V: Análise dos resultados;
- Capítulo VI: Conclusões;
- Capítulo VII: Bibliografia.

. 1. Caso de Estudo

1.1. notMnist Dataset

O caso de estudo em questão tem por um base o notMnist Dataset, um dataset desenvolvido por Yaroslav Bulatov que é uma alternativa ao dataset tradicional Mnist para reconhecimento das letras do alfabeto, sendo mais complexo. Sendo que para o trabalho usamos a sua versão mais curta que contém as suas especificações:

o Número de exemplos: 18724

o Número de atributos: Imagem 28x28 (754)

o Número de classes: 10 classes (letras A-J)

1.2.Cuckoo Search

Outro algoritmo meta-heurístico é o cuckoo search optimization(CSO) desenvolvido por Xin-She Yang and Suash Deb em 2009 e é inspirado no método em como os cucos garantem a continuidade da sua espécie. O Cuco é uma ave parasita, isto é, ao invés de construírem um ninho para colocar os seus ovos metem em ninhos de outras aves hospedeiras. (sendo que algumas espécies podem-se envolver diretamente.).Certas espécies de cuco retiram os ovos da espécie onde colocaram os seus ovos para aumentar a probabilidade de sobrevivência dos seus. As fêmeas de cuco também são especializadas no mimetismo na reprodução dos ovos para estes serem semelhantes aos dos hospedeiros. Usualmente, os cucos também inserem os seus ovos em ninhos onde as aves hospedeiras acabaram de pousar os seus ovos. Quando os pássaros hospedeiros reparam que os ovos não são seus estes retiram os ovos do Cuco do seu ninho ou então largam o ninho e constroem outro.

O CSO é baseado em 3 regras:

- Cada cuco põe um ovo que representa um conjunto de soluções coordenadas, que em determinado momento é largado em um ninho aleatório

- Uma fração dos ninhos contendo os melhores ovos ou soluções vão ser selecionados para as próximas gerações

- O número de ninhos é fixo e há uma probabilidade dos hospedeiros encontrarem um ovo de cuco(pa). Se isto acontecer, o hospedeiro pode mandar o ovo de cuco fora ou contruir um novo ninho num local diferente.

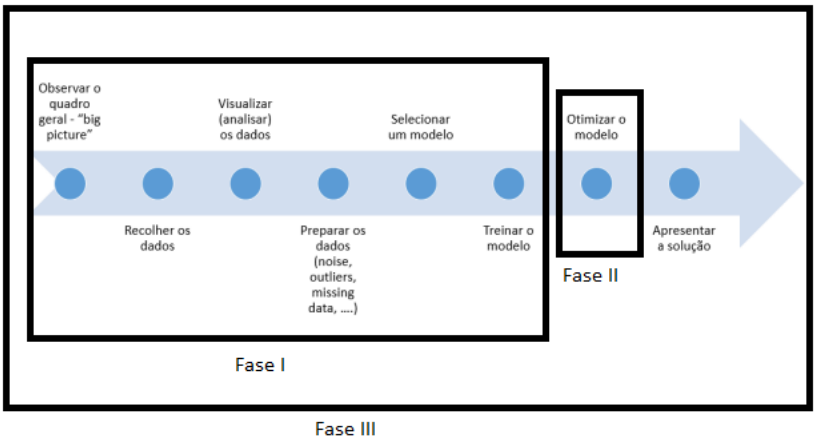

Fig. 1 – Etapas de um processo de Machine learning

2. Metodologias

2.1. NatureInspiredSearchCV

O NatureInspiredSearchCV é uma classe que tem a capacidade de usar diversos algoritmos inspirados na natureza. Esta classe é usada para otimização de Hiper parâmetros. O seu uso é semelhante ao GridSearchCV do sklearn. O algoritmo de otimização vai realizar várias execuções, nas quais otimiza a população de um determinado tamanho por um determinado número de gerações. As mesmas combinações de parâmetros estão a ser armazenadas em cache, portanto, pode ser benéfico fazer mais execuções de otimização.

```
class NatureInspiredSearchCV(estimator, param_grid, algorithm='hba', population_size=50, max_n_gen=100,
            max_stagnating_gen=20, random_state=None, scoring=None, refit=True, verbose=0,
            pre_dispatch='2*n_jobs', error_score=np.nan, return_train_score=True)
```

Fig.2 - Class NatureInspiredSearchCV e respetivos argumentos

2.2. CuckooSearch

Algoritmo usado para otimização de Hiper parâmetros, anteriormente referido e estudado. É utilizado pela metodologia NatureInspiredSearchCV, acima referida, para otimizar os Hiper parâmetros nela escolhidos. Tem como parâmetro de entrada, pa (probabilidade de o ovo ser detetado)

2.3. GridSearchCV

GridSearchCV é uma classe que tenta todas as combinações dos valores passados no dicionário e avalia o modelo para cada combinação usando o método de validação cruzada. Ou seja, após usar essa função, obtemos a accuracy para cada combinação de Hiper parâmetros e podemos escolher aquele com o melhor desempenho.

class sklearn.model_selection.GridSearchCV(*estimator, param_grid, *, scoring=None, n_jobs=None, refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', error_score=nan, return_train_score=False*) ¶ [source]

Fig.3 – Class GridSearchCV e respetivos argumentos

2.4. RandomForestClassifier

Apesar de na primeira fase do trabalho termos implementado como classificador uma rede neuronal MLP, decidimos utilizar para esta meta final, o Random Forest Classifier, um classificador que na nossa opinião é mais versátil no que toca á problemática deste Projeto. O RandonForest ou florestas de decisão aleatórias são um método de aprendizagem para problemas de para classificação e regressão. É também um dos algoritmos mais utilizados, pela sua simplicidade e diversidade. Este método constrói uma infinidade de árvores de decisão durante o treino e gera para cada class do conjunto de treino o valor da precisão.

class sklearn.ensemble.RandomForestClassifier(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None*) ¶ [source]

Fig.4 – Class RandomForestClassifier e respetivos argumentos

3. Diagrama de classes

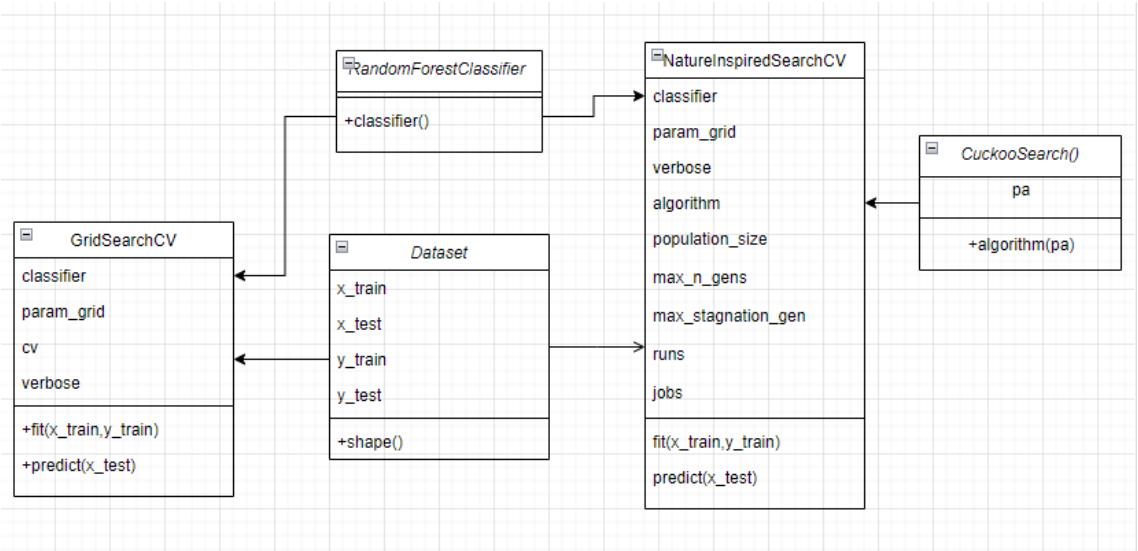

Fig.5-Diagrama de classes

Este diagrama representa como funcionam todos os algoritmos utilizados neste projeto. O Data set é composto por um conjunto de treino e um conjunto de teste que vão ser testados através dos algoritmos implementados. Ambos os algoritmos de pesquisa, O NatureInspiredCV e o GridSearchCV, utilizam como classificador o RandomForestClassifier sendo que o NatureInspiredSearchCV utiliza o algoritmo de enxame no nosso caso o CuckooSearch que apresenta como argumento pa, a probabilidade do ovo ser detetado. Estes dois algoritmos vão devolver para cada uma das 10 classes do caso de estudo a precision, o recall, o f1-score, a média de cada um deles, a accuracy e o melhor parâmetro.

4. Implementação dos algoritmos

Import Dataset

Primeiramente tivemos de importar o nosso dataset, isto é, tivemos que ler da pasta original em que ele se encontrava e passá-lo para a forma de um dataset usável.

Assim conseguimos passar de uma pasta com 10 subpastas e um total de 18724 imagens para um um set de treino [13106,10] e um set de teste [5618,10]

Cuckoo search

Para implementarmos o cuckoo search utilizamos a biblioteca Niapy que já tem este algoritmo implementado e nos permite usar na classe NatureInspiredSearchCV

```
from niapy.algorithms.basic import CuckooSearch
algorithm = CuckooSearch()
```

Fig.6-Import do algoritmo CuckooSearch

Modelo MLP

Como anteriormente foi referido optamos por implementar como classificador do modelo o RandomForestClassifier, mas apesar disso também implementamos um modelo como uma rede mlp tal como tínhamos feito na fase I deste projeto e usando as funcionalidades da bivlioteca keras exportamos para um ficheiro á parte este mesmo modelo já treinado mas não otimizado para que possa ser otimizado caso o utilizador o pretenda

RandomForestClassifier

Para importar o classificador RandomForestClassifier foi utilizado o sklearn.ensemble. O classificador contém diversos Hiper parâmetros que podem ser otimizados. De todos nós escolhemos dois:

• n_estimators – Número de árvores da floresta;

• max_depth – Profundidade da árvore;

```
from sklearn.ensemble import RandomForestClassifier

param_grid = {
    'n_estimators': range(20, 100, 20),
    'max_depth': range(2, 20, 2),
}
```

Fig.7-Hiper parâmetros a serem otimizados

NatureInspiredSearchCV

Para importar a biblioteca NatureInspiredSearchCV foi utilizado o sklearn_nature_inspired_algotithms. Utilizamos os seguintes parâmetros nos testes:

```
rom sklearn_nature_inspired_algorithms.model_selection import NatureInspiredSearchCV

ia_search = NatureInspiredSearchCV(
    clf,
    param_grid,
    cv=2,
    verbose=3,
    algorithm=algorithm,
    population_size=50,
    max_n_gen=100,
    max_stagnating_gen=20,
    runs=1
```

Fig.8-Parametros default para os testes

GridSeachCV

Para comparação de resultados usamos o algoritmo default com os seguintes parametros:

```
from sklearn.model_selection import GridSearchCV

grid_search = GridSearchCV(
    clf,
    param_grid,
    cv=2,
    verbose=3,
)
```

Fig.9-Parametros do GridSeachCV

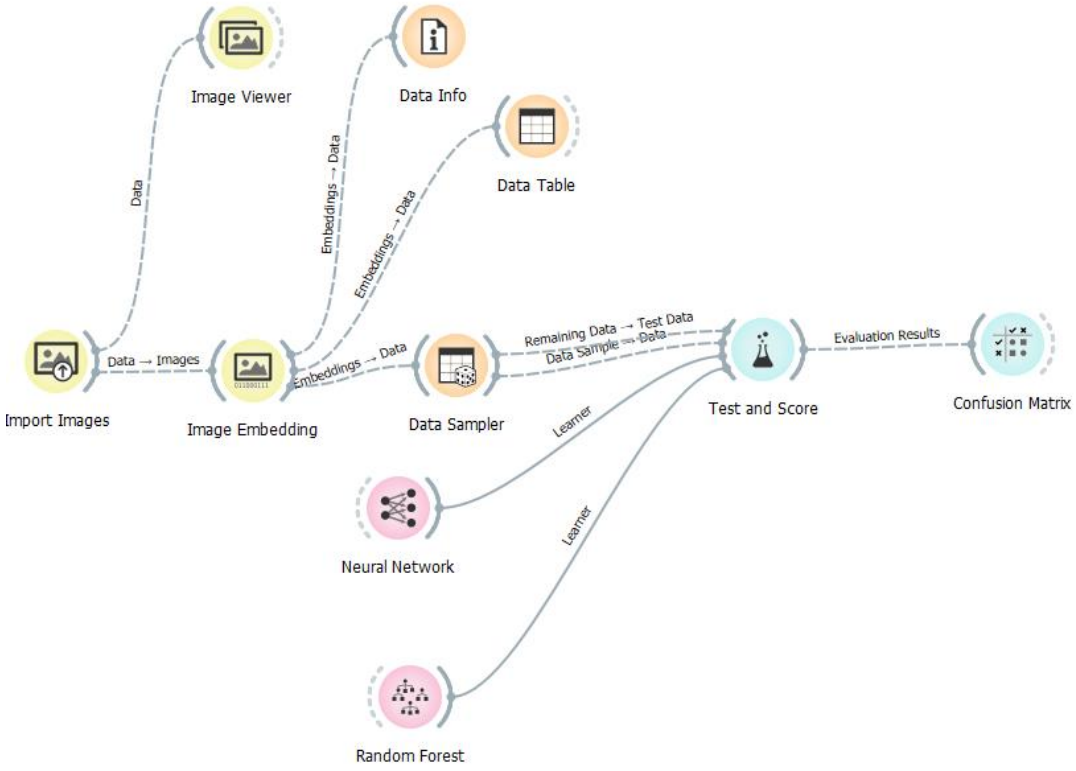

Fig.10-Workflow do Projeto no orange

5. Análise de Resultados

Tal como anteriormente referido decidimos usar como Hiper parâmetros a otimizar n_estimators – Número de árvores da floresta e max_depth – Profundidade da árvore.

Para tal usamos variamos estes dois Hiper parâmetros em range números tal como vai ser referido a seguir.

Começamos com valores aleatórios nos primeiros testes e fomos tentando aprimorar estes mesmos valores até chegar a uma melhor solução.

Teste 1:

NatureInspiredSearchCV – CuckooSearch

HiperParametros:

N_estimators (20,100,20), Max_depth(2,20,2) e pa(0.5)

	precision	recall	f1-score	support
0	0.9978	0.8053	0.8913	565
1	0.9953	0.7326	0.8440	576
2	0.9915	0.8345	0.9062	556
3	0.9977	0.7856	0.8790	555
4	0.9908	0.7688	0.8658	558
5	0.9917	0.8856	0.9357	542
6	0.9861	0.7933	0.8793	537
7	0.9978	0.7986	0.8871	561
8	0.9821	0.8003	0.8819	616
9	0.9734	0.8623	0.9145	552
micro avg	0.9902	0.8062	0.8887	5618
macro avg	0.9904	0.8067	0.8885	5618
weighted avg	0.9904	0.8062	0.8881	5618
samples avg	0.8062	0.8062	0.8062	5618

```
Melhor Parametro:
 {'max_depth': 12, 'n_estimators': 80}
Melhor accuracy:
 0.7920036624446818
```

GridSearchCV:

HiperParametros:

N_estimators (20,100,20), Max_depth(2,20,2) e pa(0.5)

	precision	recall	f1-score	support
0	0.9978	0.8053	0.8913	565
1	0.9953	0.7326	0.8440	576
2	0.9915	0.8345	0.9062	556
3	0.9977	0.7856	0.8790	555
4	0.9908	0.7688	0.8658	558
5	0.9917	0.8856	0.9357	542
6	0.9861	0.7933	0.8793	537
7	0.9978	0.7986	0.8871	561
8	0.9821	0.8003	0.8819	616
9	0.9734	0.8623	0.9145	552
micro avg	0.9902	0.8062	0.8887	5618
macro avg	0.9904	0.8067	0.8885	5618
weighted avg	0.9904	0.8062	0.8881	5618
samples avg	0.8062	0.8062	0.8062	5618

```
Melhor Parametro:
 {'max_depth': 12, 'n_estimators': 80}
Melhor accuracy:
 0.7920036624446818
```

Teste 2

NatureInspiredSearchCV – CuckooSearch

HiperParametros:

N_estimators (20,100,20), Max_depth(2,20,2) e pa(0.1)

```
Fitting 2 folds for each of 36 candidates, totalling 72 fits
              precision    recall  f1-score   support

           0     0.9978    0.8053    0.8913       565
           1     0.9953    0.7326    0.8440       576
           2     0.9915    0.8345    0.9062       556
           3     0.9977    0.7856    0.8790       555
           4     0.9908    0.7688    0.8658       558
           5     0.9917    0.8856    0.9357       542
           6     0.9861    0.7933    0.8793       537
           7     0.9978    0.7986    0.8871       561
           8     0.9821    0.8003    0.8819       616
           9     0.9734    0.8623    0.9145       552

   micro avg     0.9902    0.8062    0.8887      5618
   macro avg     0.9904    0.8067    0.8885      5618
weighted avg     0.9904    0.8062    0.8881      5618
 samples avg     0.8062    0.8062    0.8062      5618

Melhor Parametro:
 {'max_depth': 12, 'n_estimators': 80}
Melhor accuracy:
 0.7920036624446818
```

GridSearchCV:

HiperParametros:

N_estimators (20,100,20), Max_depth(2,20,2) e pa(0.1)

```
Fitting 2 folds for each of 36 candidates, totalling 72 fits
              precision    recall  f1-score   support

           0     0.9978    0.8053    0.8913       565
           1     0.9953    0.7326    0.8440       576
           2     0.9915    0.8345    0.9062       556
           3     0.9977    0.7856    0.8790       555
           4     0.9908    0.7688    0.8658       558
           5     0.9917    0.8856    0.9357       542
           6     0.9861    0.7933    0.8793       537
           7     0.9978    0.7986    0.8871       561
           8     0.9821    0.8003    0.8819       616
           9     0.9734    0.8623    0.9145       552

   micro avg     0.9902    0.8062    0.8887      5618
   macro avg     0.9904    0.8067    0.8885      5618
weighted avg     0.9904    0.8062    0.8881      5618
 samples avg     0.8062    0.8062    0.8062      5618

Melhor Parametro:
 {'max_depth': 12, 'n_estimators': 80}
Melhor accuracy:
 0.7920036624446818
```

Teste 3:

NatureInspiredSearchCV – CuckooSearch

HiperParametros:

N_estimators (100,200,20), Max_depth(20,40,2) e pa(0.5)

	precision	recall	f1-score	support
0	1.0000	0.8142	0.8976	565
1	0.9930	0.7431	0.8500	576
2	0.9895	0.8453	0.9117	556
3	0.9978	0.8018	0.8891	555
4	0.9954	0.7760	0.8721	558
5	0.9918	0.8893	0.9377	542
6	0.9838	0.7896	0.8760	537
7	0.9978	0.8057	0.8915	561
8	0.9881	0.8084	0.8893	616
9	0.9755	0.8659	0.9175	552
micro avg	0.9911	0.8135	0.8935	5618
macro avg	0.9913	0.8139	0.8933	5618
weighted avg	0.9913	0.8135	0.8930	5618
samples avg	0.8135	0.8135	0.8135	5618

```
Melhor Parametro:
 {'max_depth': 20, 'n_estimators': 180}
Melhor accuracy:
 0.7916221577903251
```

GridSearchCV:

HiperParametros:

N_estimators (100,200,20), Max_depth(20,40,2) e pa(0.5)

	precision	recall	f1-score	support
0	1.0000	0.8142	0.8976	565
1	0.9930	0.7431	0.8500	576
2	0.9895	0.8453	0.9117	556
3	0.9978	0.8018	0.8891	555
4	0.9954	0.7760	0.8721	558
5	0.9918	0.8893	0.9377	542
6	0.9838	0.7896	0.8760	537
7	0.9978	0.8057	0.8915	561
8	0.9881	0.8084	0.8893	616
9	0.9755	0.8659	0.9175	552
micro avg	0.9911	0.8135	0.8935	5618
macro avg	0.9913	0.8139	0.8933	5618
weighted avg	0.9913	0.8135	0.8930	5618
samples avg	0.8135	0.8135	0.8135	5618

```
Melhor Parametro:
 {'max_depth': 20, 'n_estimators': 180}
Melhor accuracy:
 0.7916221577903251
```

Teste 4:

NatureInspiredSearchCV – CuckooSearch

HiperParametros:

N_estimators (100,200,20), Max_depth(2,20,2) e pa(0.5)

	precision	recall	f1-score	support
0	0.9978	0.8053	0.8913	565
1	0.9929	0.7309	0.8420	576
2	0.9915	0.8381	0.9084	556
3	0.9977	0.7856	0.8790	555
4	0.9954	0.7724	0.8698	558
5	0.9897	0.8875	0.9358	542
6	0.9836	0.7821	0.8714	537
7	0.9978	0.7968	0.8860	561
8	0.9802	0.8019	0.8821	616
9	0.9736	0.8696	0.9187	552
micro avg	0.9897	0.8065	0.8888	5618
macro avg	0.9900	0.8070	0.8885	5618
weighted avg	0.9900	0.8065	0.8881	5618
samples avg	0.8065	0.8065	0.8065	5618

```
Melhor Parametro:
 {'max_depth': 12, 'n_estimators': 160}
Melhor accuracy:
 0.7925377689607813
```

GridSearchCV:

HiperParametros:

N_estimators (100,200,20), Max_depth(2,20,2) e pa(0.5)

	precision	recall	f1-score	support
0	0.9978	0.8053	0.8913	565
1	0.9929	0.7309	0.8420	576
2	0.9915	0.8381	0.9084	556
3	0.9977	0.7856	0.8790	555
4	0.9954	0.7724	0.8698	558
5	0.9897	0.8875	0.9358	542
6	0.9836	0.7821	0.8714	537
7	0.9978	0.7968	0.8860	561
8	0.9802	0.8019	0.8821	616
9	0.9736	0.8696	0.9187	552
micro avg	0.9897	0.8065	0.8888	5618
macro avg	0.9900	0.8070	0.8885	5618
weighted avg	0.9900	0.8065	0.8881	5618
samples avg	0.8065	0.8065	0.8065	5618

```
Melhor Parametro:
 {'max_depth': 12, 'n_estimators': 160}
Melhor accuracy:
 0.7925377689607813
```

Teste 5:

NatureInspiredSearchCV – CuckooSearch

HiperParametros:

N_estimators (150,190,5), Max_depth(10,14,1) e pa(0.5)

	precision	recall	f1-score	support
0	1.0000	0.8142	0.8976	565
1	0.9953	0.7344	0.8452	576
2	0.9915	0.8417	0.9105	556
3	0.9977	0.7946	0.8847	555
4	0.9977	0.7778	0.8741	558
5	0.9898	0.8930	0.9389	542
6	0.9837	0.7858	0.8737	537
7	0.9978	0.8075	0.8926	561
8	0.9860	0.8019	0.8845	616
9	0.9758	0.8750	0.9226	552
micro avg	0.9913	0.8120	0.8928	5618
macro avg	0.9915	0.8126	0.8924	5618
weighted avg	0.9915	0.8120	0.8921	5618
samples avg	0.8120	0.8120	0.8120	5618

```
Melhor Parametro:
 {'max_depth': 13, 'n_estimators': 175}
Melhor accuracy:
 0.7936822829238517
```

NatureInspiredSearchCV – CuckooSearch

HiperParametros:

N_estimators (150,190,5), Max_depth(10,14,1) e pa(0.5) e population size(100)

	precision	recall	f1-score	support
0	1.0000	0.8142	0.8976	565
1	0.9953	0.7344	0.8452	576
2	0.9915	0.8417	0.9105	556
3	0.9977	0.7946	0.8847	555
4	0.9977	0.7778	0.8741	558
5	0.9898	0.8930	0.9389	542
6	0.9837	0.7858	0.8737	537
7	0.9978	0.8075	0.8926	561
8	0.9860	0.8019	0.8845	616
9	0.9758	0.8750	0.9226	552
micro avg	0.9913	0.8120	0.8928	5618
macro avg	0.9915	0.8126	0.8924	5618
weighted avg	0.9915	0.8120	0.8921	5618
samples avg	0.8120	0.8120	0.8120	5618

```
Melhor Parametro:
 {'max_depth': 13, 'n_estimators': 175}
Melhor accuracy:
 0.7936822829238517
```

NatureInspiredSearchCV – CuckooSearch

HiperParametros:

N_estimators (150,190,5), Max_depth(10,14,1) e pa(0.1)

	precision	recall	f1-score	support
0	1.0000	0.8142	0.8976	565
1	0.9953	0.7344	0.8452	576
2	0.9915	0.8417	0.9105	556
3	0.9977	0.7946	0.8847	555
4	0.9977	0.7778	0.8741	558
5	0.9898	0.8930	0.9389	542
6	0.9837	0.7858	0.8737	537
7	0.9978	0.8075	0.8926	561
8	0.9860	0.8019	0.8845	616
9	0.9758	0.8750	0.9226	552
micro avg	0.9913	0.8120	0.8928	5618
macro avg	0.9915	0.8126	0.8924	5618
weighted avg	0.9915	0.8120	0.8921	5618
samples avg	0.8120	0.8120	0.8120	5618

```
Melhor Parametro:
 {'max_depth': 13, 'n_estimators': 175}
Melhor accuracy:
 0.7936822829238517
```

NatureInspiredSearchCV – CuckooSearch

HiperParametros:

N_estimators (150,190,5), Max_depth(10,14,1) e pa(0.2)

	precision	recall	f1-score	support
0	1.0000	0.8142	0.8976	565
1	0.9953	0.7344	0.8452	576
2	0.9915	0.8417	0.9105	556
3	0.9977	0.7946	0.8847	555
4	0.9977	0.7778	0.8741	558
5	0.9898	0.8930	0.9389	542
6	0.9837	0.7858	0.8737	537
7	0.9978	0.8075	0.8926	561
8	0.9860	0.8019	0.8845	616
9	0.9758	0.8750	0.9226	552
micro avg	0.9913	0.8120	0.8928	5618
macro avg	0.9915	0.8126	0.8924	5618
weighted avg	0.9915	0.8120	0.8921	5618
samples avg	0.8120	0.8120	0.8120	5618

```
Melhor Parametro:
 {'max_depth': 13, 'n_estimators': 175}
Melhor accuracy:
 0.7936822829238517
```

NatureInspiredSearchCV – CuckooSearch

HiperParametros:

N_estimators (150,190,5), Max_depth(10,14,1) e pa(0.75)

	precision	recall	f1-score	support
0	1.0000	0.8142	0.8976	565
1	0.9953	0.7344	0.8452	576
2	0.9915	0.8417	0.9105	556
3	0.9977	0.7946	0.8847	555
4	0.9977	0.7778	0.8741	558
5	0.9898	0.8930	0.9389	542
6	0.9837	0.7858	0.8737	537
7	0.9978	0.8075	0.8926	561
8	0.9860	0.8019	0.8845	616
9	0.9758	0.8750	0.9226	552
micro avg	0.9913	0.8120	0.8928	5618
macro avg	0.9915	0.8126	0.8924	5618
weighted avg	0.9915	0.8120	0.8921	5618
samples avg	0.8120	0.8120	0.8120	5618

```
Melhor Parametro:
 {'max_depth': 13, 'n_estimators': 175}
Melhor accuracy:
 0.7936822829238517
```

GridSearchCV:

HiperParametros:

N_estimators (150,190,5), Max_depth(10,14,1) e pa(0.5)

	precision	recall	f1-score	support
0	1.0000	0.8142	0.8976	565
1	0.9953	0.7344	0.8452	576
2	0.9915	0.8417	0.9105	556
3	0.9977	0.7946	0.8847	555
4	0.9977	0.7778	0.8741	558
5	0.9898	0.8930	0.9389	542
6	0.9837	0.7858	0.8737	537
7	0.9978	0.8075	0.8926	561
8	0.9860	0.8019	0.8845	616
9	0.9758	0.8750	0.9226	552
micro avg	0.9913	0.8120	0.8928	5618
macro avg	0.9915	0.8126	0.8924	5618
weighted avg	0.9915	0.8120	0.8921	5618
samples avg	0.8120	0.8120	0.8120	5618

```
Melhor Parametro:
 {'max_depth': 13, 'n_estimators': 175}
Melhor accuracy:
 0.7936822829238517
```

Teste 6:

NatureInspiredSearchCV – CuckooSearch

HiperParametros:

N_estimators (170,180,1), Max_depth(10,14,1) e pa(0.5)

	precision	recall	f1-score	support
0	1.0000	0.8142	0.8976	565
1	0.9953	0.7344	0.8452	576
2	0.9915	0.8417	0.9105	556
3	0.9977	0.7946	0.8847	555
4	0.9977	0.7778	0.8741	558
5	0.9898	0.8930	0.9389	542
6	0.9837	0.7858	0.8737	537
7	0.9978	0.8075	0.8926	561
8	0.9860	0.8019	0.8845	616
9	0.9758	0.8750	0.9226	552
micro avg	0.9913	0.8120	0.8928	5618
macro avg	0.9915	0.8126	0.8924	5618
weighted avg	0.9915	0.8120	0.8921	5618
samples avg	0.8120	0.8120	0.8120	5618

```
Melhor Parametro:
 {'max_depth': 13, 'n_estimators': 175}
Melhor accuracy:
 0.7936822829238517
```

GridSearchCV:

HiperParametros:

N_estimators (170,180,1), Max_depth(10,14,1) e pa(0.5)

	precision	recall	f1-score	support
0	1.0000	0.8142	0.8976	565
1	0.9953	0.7344	0.8452	576
2	0.9915	0.8417	0.9105	556
3	0.9977	0.7946	0.8847	555
4	0.9977	0.7778	0.8741	558
5	0.9898	0.8930	0.9389	542
6	0.9837	0.7858	0.8737	537
7	0.9978	0.8075	0.8926	561
8	0.9860	0.8019	0.8845	616
9	0.9758	0.8750	0.9226	552
micro avg	0.9913	0.8120	0.8928	5618
macro avg	0.9915	0.8126	0.8924	5618
weighted avg	0.9915	0.8120	0.8921	5618
samples avg	0.8120	0.8120	0.8120	5618

```
Melhor Parametro:
 {'max_depth': 13, 'n_estimators': 175}
Melhor accuracy:
 0.7936822829238517
```

6.Conclusão

Quanto a conclusões retiradas dos testes, podemos concluir através da análise dos resultados que nos testes que com valores de range iguais dos hiperparametros a optimizar obtivemos os mesmos resultados no qye diz respeito ao GridSearchCV e ao NatureInspiredSearchCV – CuckooSearch. Tendo por esta base, começamos no 1º teste por introduzir valores aleatórios e obtivemos uma accuracy inicial cerca de 79,20% e os melhores parametros foram Max_depth 12, n_estimators 80

No segundo teste para ver se o parâmetro pa do CuckooSearch tinha influencia no resultado final diminuímos este para 0.1 e obtivemos os mesmos resultados do primeiro teste.

No terceiro teste dobramos o range de valores do N_estimators e Max_depth e obtivemos uma piora da accuracy para 79,16% e os melhores parametros foram Max_depth 20, n_estimators 180

No quarto teste ao apercebermos da piora na accuracy descemos novamente o n_estimators para o valor do primeiro teste mantendo max_depth do terceiro teste e obtivemos uma melhora na accuracy para 79,25% sendo os melhores parametros Max_depth 12, n_estimators 160

No quinto parâmetro apercebemo-nos que só iriamos conseguir melhoras se diminuíssemos o passo dentro dos ranges sendo que diminuímos o passo da max_depth para 1 e o passo do n_estimators para 5 e obtivemos uma melhoria para 79,36% sendo os melhores parâmetros Max_depth 13, n_estimators 175, neste teste ainda verificamos que mudar o valor de pa não alterava os resultados obtidos.

Por último, no sexto teste diminuímos o passo do n_estimators para 1 sendo que obtivemos os mesmos valores do quinto teste chegando assim ao valor máximo que consideramos razoável.

Como podemos ver o CuckooSearch é um excelente algoritmo para otimização de Hiper parâmetros sendo que obteve os mesmos resultados de uma função benchmark que é utilizada especialmente para este mesmo propósito.

Por fim com este trabalho conseguimos aprofundar os nossos conhecimentos tanto práticos como teóricos na área de Inteligência computacional.

Referencias:

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

https://github.com/scikit-learn/scikit-learn

https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

https://sklearn-nature-inspired-algorithms.readthedocs.io/en/latest/advanced/niapy.html

https://sklearn-nature-inspired-algorithms.readthedocs.io/en/latest/introduction/nature-inspired-search-cv.html

https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/