Atividade 2: Roosevelt National Forest

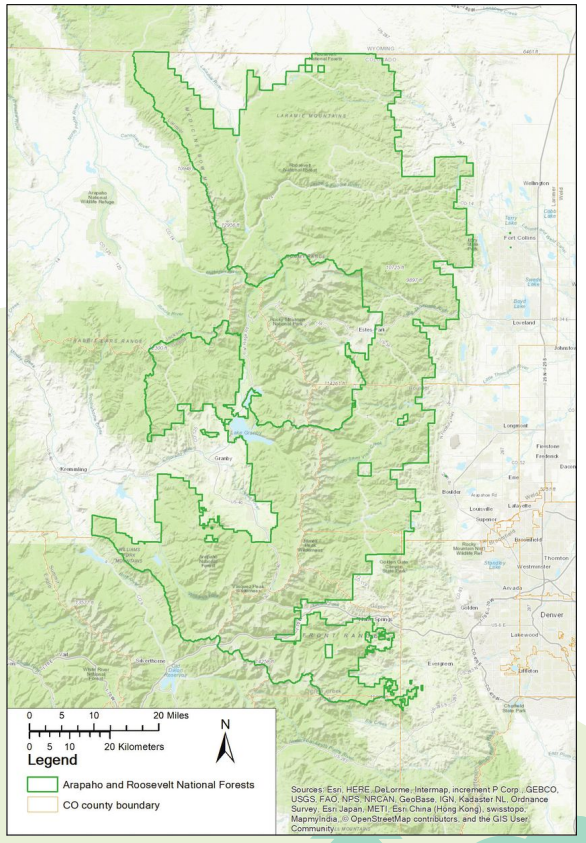
Redes Neurais Artificiais – BSI07_T01

Benjamin Borges
Delrick Oliveira
Jogno Vezú Júnior
Max Dias

Introdução

A Base de Dados que trabalhamos consiste em:
Observações de árvores de quatro áreas da Floresta Nacional Roosevelt, no Colorado.
Em 1998;

Criada por:
Jock A. Blackard;
Dr. Denis J. Dean;
Dr. Charles W. Anderson.



Introdução

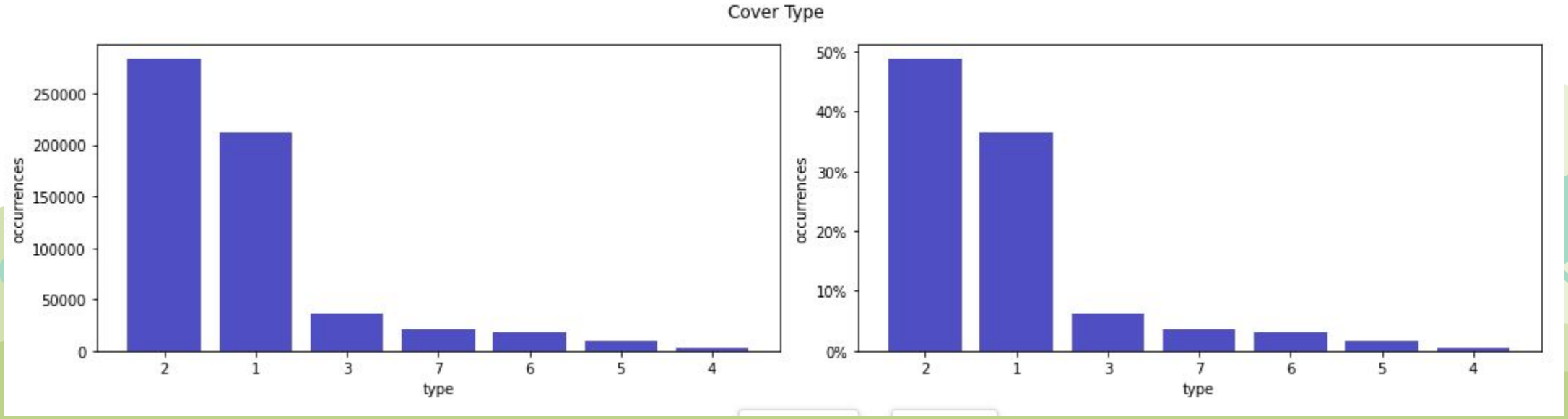
- A Base de Dados possui 581.012 amostras;
- Cada amostra apresentadas em 55 Colunas;
- Onde cada coluna apresenta informações sobre o tipo da árvore.

- Resumo das Colunas:

Elevação	Aspecto	Declive	Distância horizontal para hidrologia	Distância vertical para hidrologia	Distância horizontal para estrada	Sombra (9:00 / 12:00 / 15:00)	Distância horizontal para focos de incêndio	Área selvagem	Tipo de solo	Tipo de Cobertura

Introdução

Quanto a distribuição dos Dados:
Observa-se que, de acordo com a Base, das 581.012 amostras, mais de 80% estão na Categoria 1 e 2.
Resultando em uma base desbalanceada.



Testando Redes Neurais
Sem os atributos categóricos.

Testando Redes Neurais Sem os atributos categóricos.

Para tratarnos a base, primeiramente removemos os atributos categóricos (Area Selvagem e Tipo de Solo;

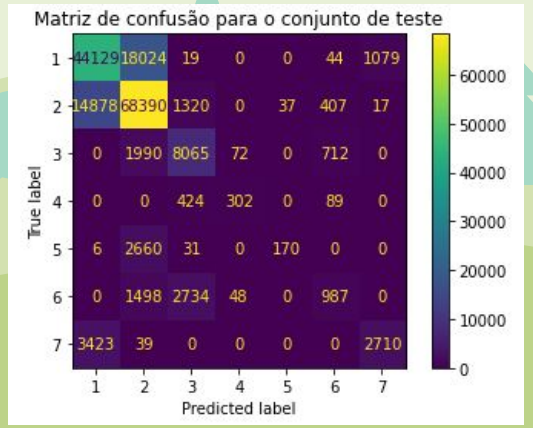
```
categoricos = list(filter(lambda x: (x.startswith('Wilderness_Area') or x.startswith('Soil_Type')), data_all.columns.to_list()))

```

Em seguida, dividos a base para Treino (70%) e para Teste (30%) e realizamos o escalonamento dos Dados;

```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, shuffle=True)

X_train_std = (X_train - np.mean(X_train))/np.std(X_train)
X_test_std = (X_test - np.mean(X_train))/np.std(X_train)
```

E realizamos o treinamento;

```
mlp = MLPClassifier(hidden_layer_sizes=(10,),
                    max_iter=300,
                    activation='relu',
                    solver='adam',
                    verbose=True,
                    random_state=1)

mlp.fit(X_train_std, y_train.values.ravel())

```




```
Acurácia - Média:  0.7152962639985313
Acurácia - Desvio Padrão :  0.00238822397143433
F1-Score - Média:  0.5203633874934127
F1-Score - Desvio Padrão:  0.01141224018916451
```

Testando Redes Neurais Sem os atributos categóricos.

Continuando com o teste, utilizamos o otimizador SGD (demais parâmetros permaneceram iguais);

```
22 ▼ mlp = MLPClassifier(hidden_layer_sizes=(10,),
23                       max_iter=300,
24                       activation='relu',
25                       solver='sgd',
26                       verbose=True,
27                       random_state=1)
28
29   mlp.fit(X_train_std, y_train.values.ravel())
```

```
Acurácia - Média:  0.7160329080227649
Acurácia - Desvio Padrão :  0.0016266547641324695
F1-Score - Média:  0.46468009813821104
F1-Score - Desvio Padrão:  0.011731148839183107
```

OBSERVAÇÃO:
Com otimizador SGD o tempo de treinamento foi maior e o F-Score foi menor em relação ao otimizador ADAM. Em termos de Acurácia, os resultados são semelhantes com ambos os otimizadores.

Nos testes realizados, usamos:: Uma única camada oculta com 10 neurônios e função de ativação ReLU, além do escalonamento dos Dados.. E o otimizador utilizado, quer seja SGD ou ADAM, trata-se do algoritmo para aproximar o gradiente;

Ao realizar o treinamento 15 vezes, faz com que eliminemos algum viés introduzido por uma boa ou má "sorte" na escolha de pesos no caso de uma única execução.

Propondo Novas Arquiteturas

Propondo Novas Arquiteturas

- **Terminado os testes com os diferente otimizadores, passamos para a fase de propor arquiteturas. As arquiteturas de camadas ocultas que propomos nesta atividades são essas.**

```
hidden_layer_sizes_list = [
                    (5,),(10,),(15,),
                    (5,5),(10,10),(15,15),
                    (10,5),(15,5),(15,10),
                    (5,10)
                    ]
```

Propondo Novas Arquiteturas

- **Efetuamos o teste com todas as combinações disponíveis entre os parâmetros camada oculta, função de ativação, e hiper parametros otimizadores; número de épocas propostos pela atividade.**

```
solvers = ['adam', 'sgd']
max_iters = [100, 150, 200]

param_keys = ['hidden_layer_sizes', 'activation', 'solver', 'max_iter']

base_params = {'random_state': 1, 'verbose': False}

results = []
for hidden_layer_sizes in hidden_layer_sizes_list:
  params = base_params.copy()
  params['hidden_layer_sizes'] = hidden_layer_sizes
  params['activation'] = 'relu'
  for s in solvers:
    params['solver'] = s
    for i in max_iters:
      params['max_iter'] = i
      print(params)
      mlp_params_all, acc_mean, acc_std, f1s_mean, f1s_std = evaluate_mlp(X, y, N=N, **params)
      summarized_mlp_params = { param_key: mlp_params_all[param_key] for param_key in param_keys }
      results.append((summarized_mlp_params.values(), acc_mean, acc_std, f1s_mean, f1s_std))
```

Propondo Novas Arquiteturas

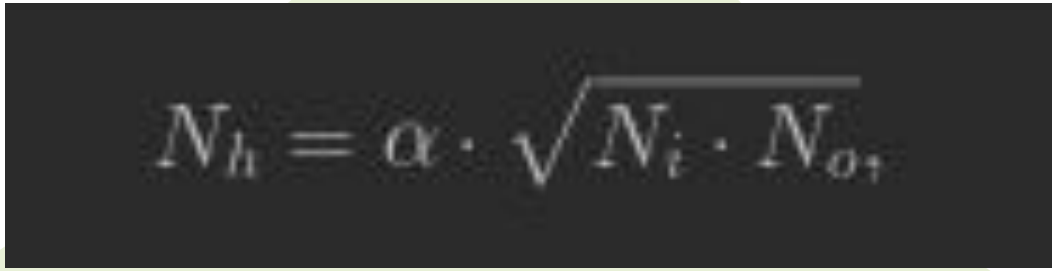
- **Após a execução as 3 arquiteturas com maior valor de acurácia e f1-score são apresentadas abaixo:**

	Configuração MLP ['hidden_layer_sizes', 'activation', 'so'	Acurácia Média	Desvio Padrão	F1-Score Média	Desvio Padrão
0	[(15, 15), relu, adam, 200]	0.753691	0.00254848	0.613159	0.0133949
1	[(15, 15), relu, adam, 150]	0.752129	0.00423977	0.607575	0.0143259
2	[(15, 15), relu, adam, 100]	0.751287	0.00230428	0.606962	0.00931045

Estimando o número de neurônios

Estimando o número de neurônios

- Após teste e análise das arquiteturas propostas, foi realizado o cálculo de novas arquiteturas a partir da regra da pirâmide geometrica, com os valores de alfa propostos pela atividade. Seguindo a seguinte fórmula:



No qual:

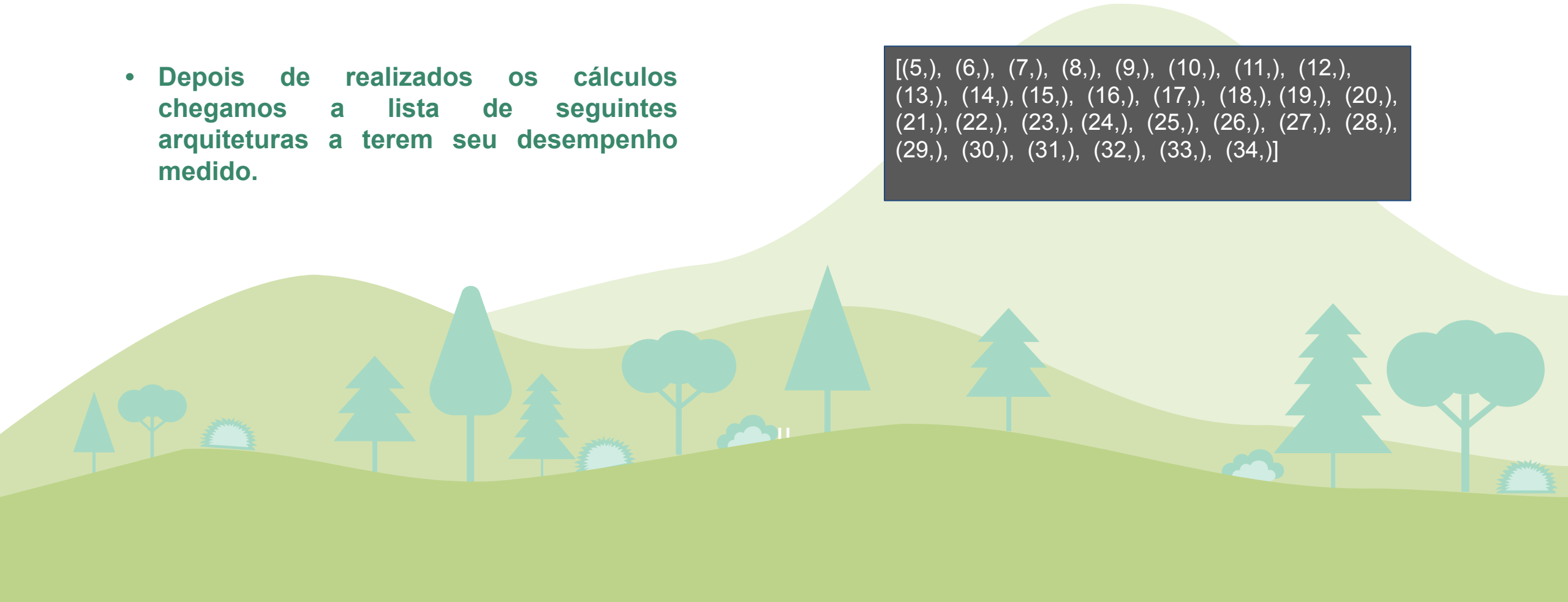
- NH é o número de neurônios ocultos a serem distribuídos em uma ou duas camadas ocultas
- Ni é o número de neurônio na camada de entrada
- No é o número de neurônios na camada de saída

```
hidden_layer_sizes_list = []
a=.5
for h in range (0,30):
  nh = a*((10*7)**(1/2))
  hidden_layer_sizes_list.append((mt.ceil(nh),))
  a+=.12
  #print(a)
hidden_layer_sizes_list
```

Estimando o número de neurônios

- **Depois de realizados os cálculos chegamos a lista de seguintes arquiteturas a terem seu desempenho medido.**

```
[(5,), (6,), (7,), (8,), (9,), (10,), (11,), (12,),
(13,), (14,), (15,), (16,), (17,), (18,), (19,), (20,),
(21,), (22,), (23,), (24,), (25,), (26,), (27,), (28,),
(29,), (30,), (31,), (32,), (33,), (34,)]
```



Estimando o número de neurônios

- **Por restrições de hardware testamos as arquiteturas somente com a quantidade de épocas 200 e somente com o otimizador adam e função de ativação relu.**

```
solvers = ['adam']
max_iters = [200]

param_keys = ['hidden_layer_sizes', 'activation', 'solver', 'max_iter']

base_params = {'verbose': False}

results = []
for hidden_layer_sizes in hidden_layer_sizes_list:
  params = base_params.copy()
  params['hidden_layer_sizes'] = hidden_layer_sizes
  params['activation'] = 'relu'
  for s in solvers:
    params['solver'] = s
    for i in max_iters:
      params['max_iter'] = i
      print(params)
      #mlp_params_all, acc_mean, acc_std, f1s_mean, f1s_std = evaluate_mlp(X, y, N=N, **params)
      #summarized_mlp_params = { param_key: mlp_params_all[param_key] for param_key in param_keys }
      #results.append((summarized_mlp_params.values(), acc_mean, acc_std, f1s_mean, f1s_std))
```

Estimando o número de neurônios

- Tanto na ordenação pela acurácia quanto na ordenação pelo f1-score seguiu-se o mesmo top 3. Para as 3 arquiteturas ao lado

```
df = pd.DataFrame(results, columns=['conf', 'acc_m', 'acc_sd', 'f1s_m', 'f1s_sd'])
```

```
df2 = df.sort_values(['acc_m', 'acc_sd'], ascending = [False, True])
df2 = df2.head(3)
df2
```

	conf	acc_m	acc_sd	f1s_m	f1s_sd
5	((34,), relu, adam, 200)	0.754082	0.001489	0.615082	0.006465
4	((28,), relu, adam, 200)	0.746168	0.004418	0.587385	0.006226
3	((23,), relu, adam, 200)	0.742398	0.002794	0.581322	0.001806

3 rows × 5 columns Open in new tab

```
df3 = df.sort_values(['f1s_m', 'f1s_sd'], ascending = [False, True])
df3 = df3.head(3)
df3
```

	conf	acc_m	acc_sd	f1s_m	f1s_sd
5	((34,), relu, adam, 200)	0.754082	0.001489	0.615082	0.006465
4	((28,), relu, adam, 200)	0.746168	0.004418	0.587385	0.006226
3	((23,), relu, adam, 200)	0.742398	0.002794	0.581322	0.001806

Testando as redes neurais com atributos categóricos

Testando as redes neurais com atributos categóricos

- Terminados os testes para propor novas arquiteturas e estimar o número de neurônios, passou a ser utilizado um dataframe que contivesse todos os atributos presentes no dataset, incluso os categóricos que haviam sido deixados de fora anteriormente.

```
data = data_all
```

```
data.shape
```

```
(581012, 55)
```

```
data.dtypes
```

```
Horizontal_Distance_To_Fire_Points    int64
Wilderness_Area1                      int64
Wilderness_Area2                      int64
Wilderness_Area3                      int64
Wilderness_Area4                      int64
Soil_Type1                            int64
Soil_Type2                            int64
Soil_Type3                            int64
Soil_Type4                            int64
Soil_Type5                            int64
Soil_Type6                            int64
Soil_Type7                            int64
Soil_Type8                            int64
Soil_Type9                            int64
Soil_Type10                           int64
```

Testando as redes neurais com atributos categóricos

- Após isso é feito o treinamento com a arquitetura que obteve melhor desempenho dentre as arquiteturas propostas ((15,15) com solver adam, ativação relu e máximo de épocas em 200).

```
X = data.iloc[:, :(data.shape[1] - 1)]
y = data.iloc[:, -1:]
```

```
#arquitetura própria
```

```
ptop1_custom = {'verbose': False, 'hidden_layer_sizes': (15,15), 'activation': 'relu', 'solver': 'adam', 'max_iter': 200}
```

```
mlp_params_all, acc_mean, acc_std, f1s_mean, f1s_std = evaluate_mlp(X, y, N=2, **ptop1_custom)
```

Testando as redes neurais com atributos categóricos

- **Comparando os resultados obtivemos uma melhora com relação ao treinamento sem os atributos categóricos, tanto na acurácia quanto com relação ao f1 score.**

Sem os atributos categóricos

	Configuração MLP ['hidden_layer_sizes', 'activation', 'so'	Acurácia Média	Desvio Padrão	F1-Score Média	Desvio Padrão
0	[(15, 15), relu, adam, 200]	0.753691	0.00254848	0.613159	0.0133949

Com os atributos categóricos

```
Acurácia - Média:  0.7973769965118414
Acurácia - Desvio Padrão :  0.0025702221406278536
F1-Score - Média:  0.6795517374981748
F1-Score - Desvio Padrão:  0.0008670632605654593
```

Testando as redes neurais com atributos categóricos

- O mesmo treinamento foi repetido com a arquitetura que foi estimada utilizando a regra da pirâmide geométrica que obteve melhor desempenho((34,0) com solver adam, ativação relu e máximo de épocas em 200).

```
#regra da pirâmide geométricaX

ptop1_geo = {'verbose': False, 'hidden_layer_sizes': (34,), 'activation': 'relu', 'solver': 'adam', 'max_iter': 200}

mlp_params_all, acc_mean, acc_std, f1s_mean, f1s_std = evaluate_mlp(X, y, N=2, **ptop1_geo)
```

Testando as redes neurais com atributos categóricos

- Comparando os resultados obtivemos uma melhora igualmente com relação à acurácia e ao f1 score.

Sem os atributos categóricos

	Configuração MLP ['hidden_layer_sizes', 'activation', 'so'	Acurácia Média	Desvio Padrão	F1-Score Média	Desvio Padrão
0	((34,), relu, adam, 200)	0.754082	0.00148878	0.615082	0.00646535

Com os atributos categóricos

```
Acurácia - Média:  0.8157156462272811
Acurácia - Desvio Padrão :  0.001414195887644576
F1-Score - Média:  0.7186963825624235
F1-Score - Desvio Padrão:  0.007296691637514063
```

Testando as redes neurais com atributos categóricos

- **Chegamos à conclusão preliminar de que neste dataset a inserção de atributos categóricos no treinamento melhora a performance do modelo, por conta das informações contidas nestes atributos ajudarem a descrever melhor cada instância do problema, contribuindo para uma melhor classificação. E também que a melhor arquitetura presente nesta parte 2 da atividade 2.2 foi a de (34,0).**

BUSCA EM GRADE

TOP 6 ARQUITETURAS

Os 2 TOPs 3 Sem Atributos Categóricos - 15 Iterações em Cada Modelo

	Configuração MLP	Acurácia		F1-Score	
	['hidden_layer_sizes', 'activation', 'solver', 'max_iter']	Média	Desvio Padrão	Média	Desvio Padrão
0	((34,), relu, adam, 200)	0.754082	0.00148878	0.615082	0.00646535
1	((28,), relu, adam, 200)	0.746168	0.00441757	0.587385	0.00622565
2	((23,), relu, adam, 200)	0.742398	0.00279397	0.581322	0.00180623

	Configuração MLP	Acurácia		F1-Score	
	['hidden_layer_sizes', 'activation', 'solver', 'max_iter']	Média	Desvio Padrão	Média	Desvio Padrão
0	[(15, 15), relu, adam, 200]	0.753691	0.00254848	0.613159	0.0133949
1	[(15, 15), relu, adam, 150]	0.752129	0.00423977	0.607575	0.0143259
2	[(15, 15), relu, adam, 100]	0.751287	0.00230428	0.606962	0.00931045

15 REPETIÇÕES

Contents

Incluindo dados sem atributos categoricos

```
[27]
data_all = pd.read_csv('data/covtype.csv')
# Elimine todas as colunas relativas aos atributos categóricos
categoricos = list(filter(lambda x: (x.startswith('Wilderness_Area') or x.startswith('Soil_Type')), data_all.columns.to_list()))
data = data_all.drop(categoricos, axis=1)
X = data.iloc[:, :(data.shape[1] - 1)]
y = data.iloc[:, -1:]

X = np.array(X)
y = np.array(y)
✓ 0.9s
```

```
[28]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
✓ 0.1s
```

Configuração

```
hiperparams = {'solver': ['adam'],
               'batch_size': [1000],
               'learning_rate_init': [0.001, 0.01],
               'max_iter': [350],
               'n_iter_no_change': [10]}

scores = {'F-Score': make_scorer(f1_score , average='weighted'),
          'Accuracy': make_scorer(accuracy_score)}
```

[29] ✓ 0.7s

```
def gridsearch(architecture):

    #Escalonamento dos atributos
    X_train_std = (X_train - np.mean(X_train))/np.std(X_train)

    model_gs = MLPClassifier(hidden_layer_sizes=architecture[0], activation=architecture[1])
    clf = GridSearchCV(model_gs, hiperparams, scoring=scores, n_jobs=-1, refit='F-Score', cv=5, verbose=160)
    clf.fit(X_train_std, y_train)

    return clf
```

[30] ✓ 0.4s

Busca em Grade

```
clf1 = gridsearch([(15,15), 'relu'])
```

✓ 15m 15.7s

```
GridSearch...

Fitting 5 folds for each of 2 candidates, totalling 10 fits
```

```
print("-----Arquitetura 1---------")
print("([(15,15), 'relu'])\n")
print("Melhor Optmização de Hiperparâmetros")
print(clf1.best_params_)
print("\nMedia F1-Score")
print(clf1.best_score_)
```

✓ 0.2s

```
-----Arquitetura 1---------
([(15,15), 'relu'])

Melhor Optmização de Hiperparâmetros
{'batch_size': 1024, 'learning_rate_init': 0.001, 'max_iter': 350, 'n_iter_no_change': 10, 'solver': 'adam'}

Media F1-Score
0.7230186134140594
```

Busca em Grade

```
...  -----Arquitetura 2---------
     ([(34,), 'relu'])

     Melhor Optmização de Hiperparâmetros
     {'batch_size': 1024, 'learning_rate_init': 0.01, 'max_iter': 350, 'n_iter_no_change': 10, 'solver': 'adam'}

     Media F1-Score
     0.7279188737056288
```

```
-----Arquitetura 1---------
([(15,15), 'relu'])

Melhor Optmização de Hiperparâmetros
{'batch_size': 1024, 'learning_rate_init': 0.001, 'max_iter': 350, 'n_iter_no_change': 10, 'solver': 'adam'}

Media F1-Score
0.7230186134140594
```

Busca em Grade

```
...  -----Arquitetura 4---------
     ([(28,), 'relu'])

     Melhor Optmização de Hiperparâmetros
     {'batch_size': 1024, 'learning_rate_init': 0.01, 'max_iter': 350, 'n_iter_no_change': 10, 'solver': 'adam'}

     Media F1-Score
     0.7180377665344146
```

```
...  -----Arquitetura 3---------
     ([(23,), 'relu'])

     Melhor Optmização de Hiperparâmetros
     {'batch_size': 1024, 'learning_rate_init': 0.01, 'max_iter': 350, 'n_iter_no_change': 10, 'solver': 'adam'}

     Media F1-Score
     0.7153785748743549
```

Incluindo dados com atributos categoricos

```
data = pd.read_csv('/content/drive/MyDrive/csv/covtype.csv')
# Elimine todas as colunas relativas aos atributos categóricos
# categoricos = list(filter(lambda x: (x.startswith('Wilderness_Area') or x.startswith('Soil_Type')), data_all.columns.to_list()))
# data = data_all.drop(categoricos, axis=1)
X = data.iloc[:, :(data.shape[1] - 1)]
y = data.iloc[:, -1:]



X = np.array(X)
y = np.array(y)
```

```
1  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
✓ 0.1s
```

Busca em Grade (com atributos categóricos)

```
-----Arquitetura 2---------
([(34,), 'relu'])

Melhor Optmização de Hiperparâmetros
{'batch_size': 1024, 'learning_rate_init': 0.001, 'max_iter': 350, 'n_iter_no_change': 10, 'solver': 'adam'}

Media F1-Score
0.7253857928393421
```

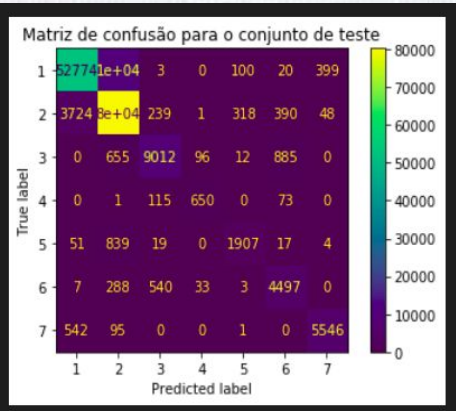
```
-----Arquitetura 4---------
([(28,), 'relu'])

Melhor Optmização de Hiperparâmetros
{'batch_size': 1024, 'learning_rate_init': 0.001, 'max_iter': 350, 'n_iter_no_change': 10, 'solver': 'adam'}

Media F1-Score
0.7222001020330407
```

Busca em Grade (com atributos categoricos)

```
-----Arquitetura 1---------
([(15,15), 'relu'])

Melhor Optmização de Hiperparâmetros
{'batch_size': 1024, 'learning_rate_init': 0.01, 'max_iter': 350, 'n_iter_no_change': 10, 'solver': 'adam'}

Media F1-Score
0.7199636256690877
```

```
-----Arquitetura 3---------
([(23,), 'relu'])

Melhor Optmização de Hiperparâmetros
{'batch_size': 1024, 'learning_rate_init': 0.001, 'max_iter': 350, 'n_iter_no_change': 10, 'solver': 'adam'}

Media F1-Score
0.7147586620721607
```

Força Bruta (sem atributos categoricos)



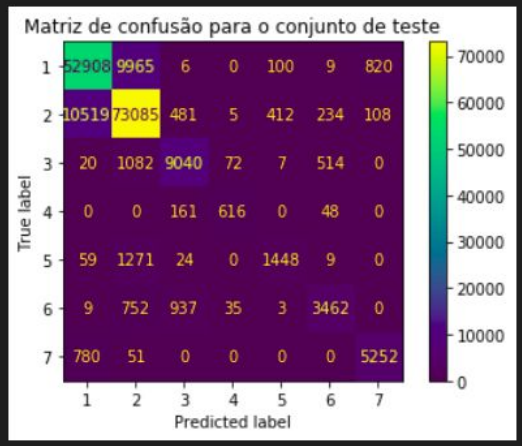

```
print('Acurácia:', accuracy_score(y_pred, y_test))
print('F1-Score:', f1_score(y_pred,y_test, average='weighted'))
print('Precisão:', precision_score(y_pred, y_test, average='weighted'))
print('Revocação:', recall_score(y_pred, y_test, average='weighted'))
```

12] ✓ 0.2s

```
Acurácia: 0.8873806682577565
F1-Score: 0.8880837428263018
Precisão: 0.892426540899862
Revocação: 0.8873806682577565
```

```
y_pred =  brute_force.predict(X_test_std)
matrix = ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.title('Matriz de confusão para o conjunto de teste')
plt.show(matrix)
plt.show()
```

✓ 3.8s



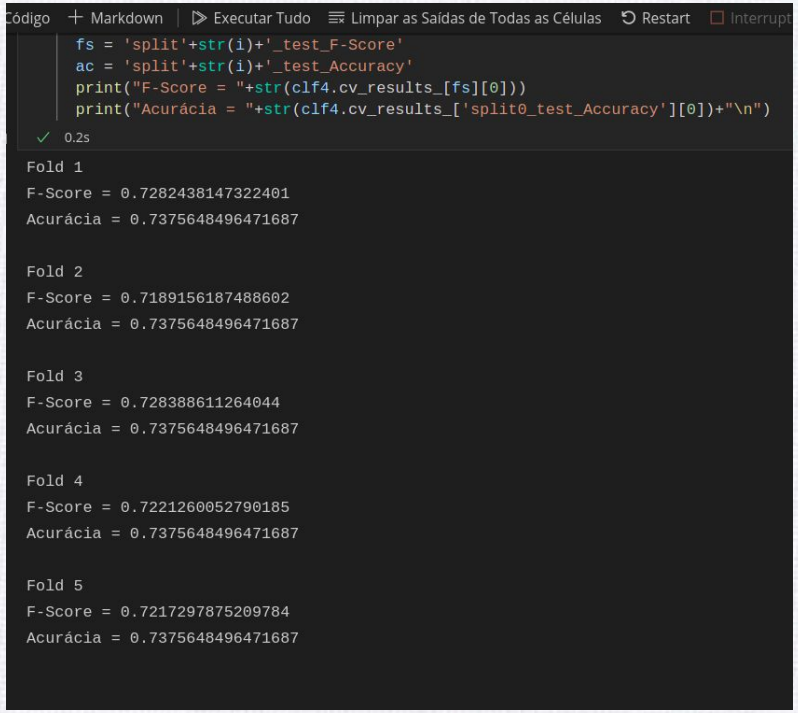

```
print('Acurácia:', accuracy_score(y_pred, y_test))
print('F1-Score:', f1_score(y_pred,y_test, average='weighted'))
print('Precisão:', precision_score(y_pred, y_test, average='weighted'))
print('Revocação:', recall_score(y_pred, y_test, average='weighted'))
```

✓ 0.2s

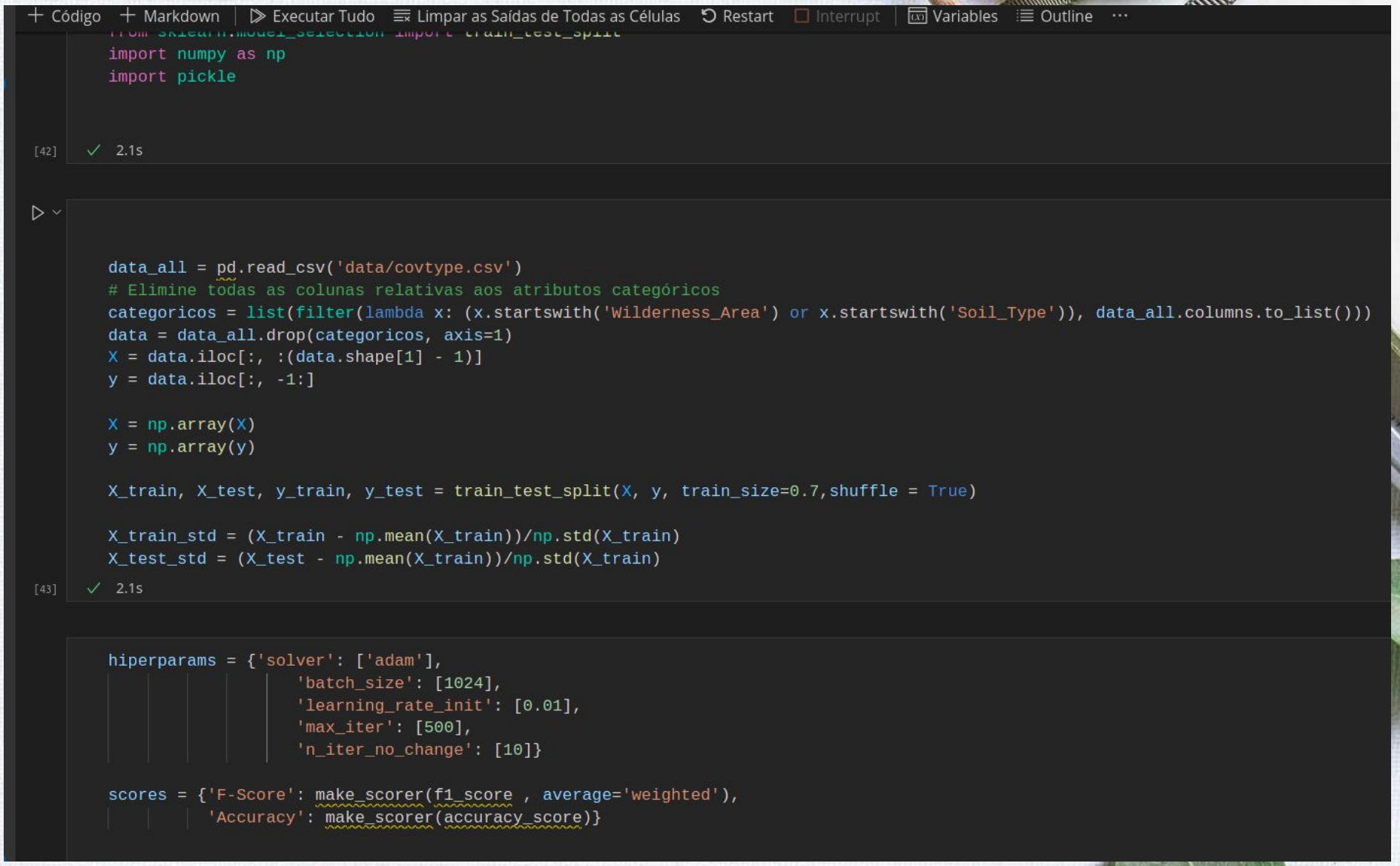
```
Acurácia: 0.8365327244354691
F1-Score: 0.8374401369328
Precisão: 0.8392214267085112
Revocação: 0.8365327244354691
```

Identificando Melhor Arquitetura

```
...  -----Arquitetura 2---------
     ([(34,), 'relu'])

     Melhor Optmização de Hiperparâmetros
     {'batch_size': 1024, 'learning_rate_init': 0.01, 'max_iter': 350, 'n_iter_no_change': 10, 'solver': 'adam'}

     Media F1-Score
     0.7279188737056288
```



EMPACOTANDO A SOLUÇÃO



EMPACOTANDO A SOLUÇÃO

```
best_network = gridsearch([(34,), 'relu'])
```

8m 7.5s

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

```
filename = 'bestnetwork.sav'
pickle.dump(best_network, open(filename, 'wb'))
```

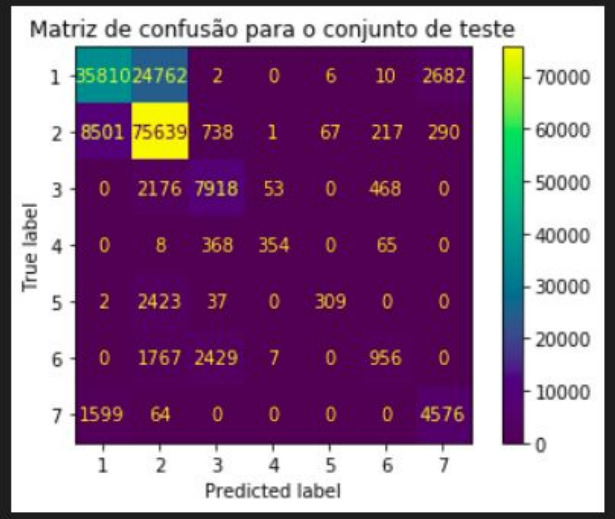
```
def load_cfl():
    pickled_best_network = pickle.load(open('bestnetwork.sav', 'rb'))
    return pickled_best_network
```

Recuperando o Modelo

```
load_best_cfl =  load_cfl()
```
✓ 0.3s

```
y_pred =  load_best_cfl.predict(X_test_std)
```
✓ 0.2s

```
matrix = ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.title('Matriz de confusão para o conjunto de teste')
plt.show(matrix)
plt.show()
```
✓ 1.2s




```
print('Acurácia:', accuracy_score(y_pred, y_test))
print('F1-Score:', f1_score(y_pred,y_test, average='weighted'))
print('Precisão:', precision_score(y_pred, y_test, average='weighted'))
print('Revocação:', recall_score(y_pred, y_test, average='weighted'))
```
✓ 0.2s

```
Acurácia: 0.7203621259408849
F1-Score: 0.7355258372779251
Precisão: 0.7756982284441815
Revocação: 0.7203621259408849
```