Insper | Instituto de Ensino e Pesquisa

Relatório do Projeto Final de Ciência dos dados

Integrantes do grupo:

Fernando Sanches

Maurício Hiroki

Theo Barbara

Objetivo do projeto

O objetivo deste projeto é aprender a utilizar técnicas de “Machine Learning”, utilizando regressão linear, classificador de Naive Bayes ou Clusterização. No caso do nosso grupo, a análise foi feita com base em um dataset (conjunto de dados) de pokémons, com o objetivo de responder à seguinte pergunta: dado um pokémon adversário, quais são os melhores pokémons para a batalha? A análise é sobre grupos de pokémons e análise dos mesmos, isto é, vamos estudar o comportamento “natural” de agrupamento dos pokémons para identificar quais são parecidos com outros. Desta forma, podemos encontrar o grupo de pokémons mais fortes para uma batalha, em que os pokémons deste grupo são parecidos e esse seria o agrupamento. Logo, a melhor ferramenta para fazer tal análise é a clusterização.

Análise exploratória dos dados

Antes de fazermos a análise, vale ressaltar que fizemos uma consideração inicial. Consideramos que todos os pokémons só terão ataques (poderes) apenas do tipo dele. Por exemplo, um pokémon tipo fogo possui apenas poderes do tipo fogo.

Nossa análise começou com a leitura do arquivo “Pokémon.csv” retirado do site kaggle, que possui 800 pokémons registrados. Nele, há informações sobre os status do pokémon (HP, Attack, Defense, Sp. Atk, Sp. Def e Speed), assim como o número do pokémon no pokedex, se ele é um Pokémon lendário ou não, sua geração e a soma de todo o status para cada pokémon, sinalizado por “Total”. A tabela a seguir, é um exemplo de pokémons mostrados com seus status numa tabela.

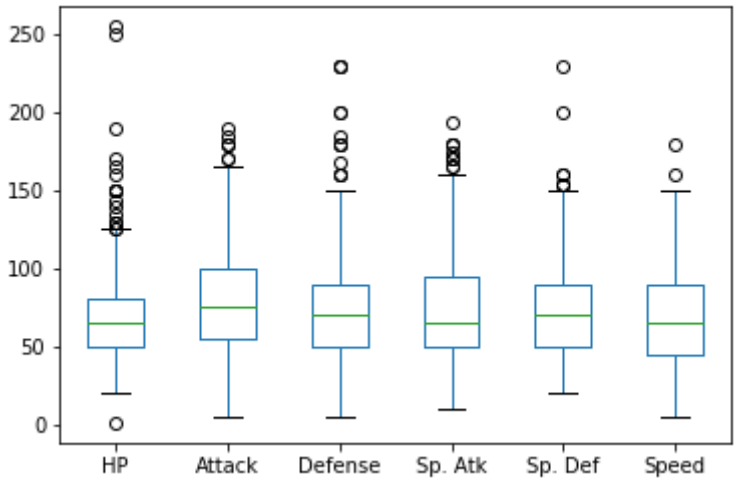
	#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
435	390	Chimchar	Fire	NaN	309	44	58	44	58	44	61	4	False
437	392	Infernape	Fire	Fighting	534	76	104	71	104	71	108	4	False
541	484	Palkia	Water	Dragon	680	90	120	100	150	120	100	4	True
549	491	Darkrai	Dark	NaN	600	70	90	90	135	90	125	4	True
139	129	Magikarp	Water	NaN	200	20	10	55	15	20	80	1	False
158	146	Moltres	Fire	Flying	580	90	100	90	125	85	90	1	True

Retirado do documento P3 jupyter notebook

Antes de analisar o conjunto inteiro de dados, formado por dados de 800 pokémons, foram separados 10 pokémons para teste. Separar pokémons para teste é importante para validar, ou melhor, testar a eficácia do resultado obtido no final do projeto. Em outras palavras, a análise dos dados por clusterização trará um resultado

obtido através da análise dos diferentes agrupamentos. Para validar, ou ainda, dizer se o agrupamento está bom ou não, é que usamos a parte teste.

Previamente, havíamos colocado um histograma com todas as variáveis do dataset. Porém, por decisão conjunta do grupo, foi decidido retirá-lo pois ficava um conjunto aglomerado de dados, difícil de compreender. Por causa disso, foi plotado um boxplot (análise exploratória) das variáveis de cada pokémon, que consistem em HP, Attack, Defense, Speed, Sp. Atk e Sp. Def:



Retirado do documento P3 jupyter notebook

Note que retiramos da análise algumas variáveis, como “#”, “Generation”, “Total” e “Legendary”. Como nosso objetivo é observar o comportamento dos 6 atributos dos status, apenas, “Generation”, “Legendary”, “#” e “Total” acabam sendo irrelevantes para tal análise. Além disso, saber se um pokémon é “Lendário”, ou saber seu número no pokedex ou ainda saber sua geração, não nos ajudará em descobrir se um pokémon é forte ou não. Esse foi outro motivo para serem retirados.

Foi verificado que muitos pokémons estão dentro dos interquartis e alguns poucos estão em outliers, considerando que há muitos pokémons. Assim, foi possível identificar que outliers, na verdade, são pokémons excepcionais, ou seja, muito fortes.

Análise dos dados por clusterização

Tendo de início, uma ideia bem geral do comportamento dos status dos pokémons, observado no boxplot, é interessante observar esses dados através de clusters. Ou seja, analisar como os pokémons são agrupados, de acordo com seus respectivos status; isto é, como podemos dividir os pokémons parecidos em grupos diferentes.

Dessa forma, é possível separar pokémons em diversos grupos. Fazendo a clusterização dos dados, algumas “saídas” foram geradas, sendo que há 11 delas (de 0 a 10). Essas saídas são parâmetros de separação dos pokémons que, em outras palavras, pode-se dizer que pokémons com saída 0, por exemplo, possuem semelhanças (são parecidos) e, portanto, pertencem ao mesmo grupo. A tabela a seguir mostra um pouco sobre o resultado desse agrupamento:

	Name	Type 1	Type 2	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	saida
0	Bulbasaur	Grass	Poison	45	49	49	65	65	45	4
1	Ivysaur	Grass	Poison	60	62	63	80	80	60	0
2	Venusaur	Grass	Poison	80	82	83	100	100	80	10
3	VenusaurMega Venusaur	Grass	Poison	80	100	123	122	120	80	10
4	Charmander	Fire	NaN	39	52	43	60	50	65	4
5	Charmeleon	Fire	NaN	58	64	58	80	65	80	0
6	Charizard	Fire	Flying	78	84	78	109	85	100	5
7	CharizardMega Charizard X	Fire	Dragon	78	130	111	130	85	100	7
8	CharizardMega Charizard Y	Fire	Flying	78	104	78	159	115	100	7
9	Squirtle	Water	NaN	44	48	65	50	64	43	4
10	Wartortle	Water	NaN	59	63	80	65	80	58	0

Retirado do documento P3 jupyter notebook

No entanto, o grupo teve que enfrentar um problema: durante os testes para clusterizar os pokémons, foi observado que a cada vez que o código era rodado, diferentes saídas eram geradas para cada pokémon.

```
base = cluster.KMeans(n_clusters=11, init='k-means++', max_iter=100, n_init=1, verbose=0, random_state=3425)

base.fit(dados_entrada)
saida_treinamento = base.predict(dados_entrada)
pokedex['saida'] = saida_treinamento
```

Retirado do documento P3 jupyter notebook

O código acima mostra, basicamente, quantos clusters queremos analisar, representado por n_clusters na “base”. Em seguida, o “.fit” seria o código gerador dos clusters e “.predict”, o responsável por classificar as saídas dos pokémons de acordo

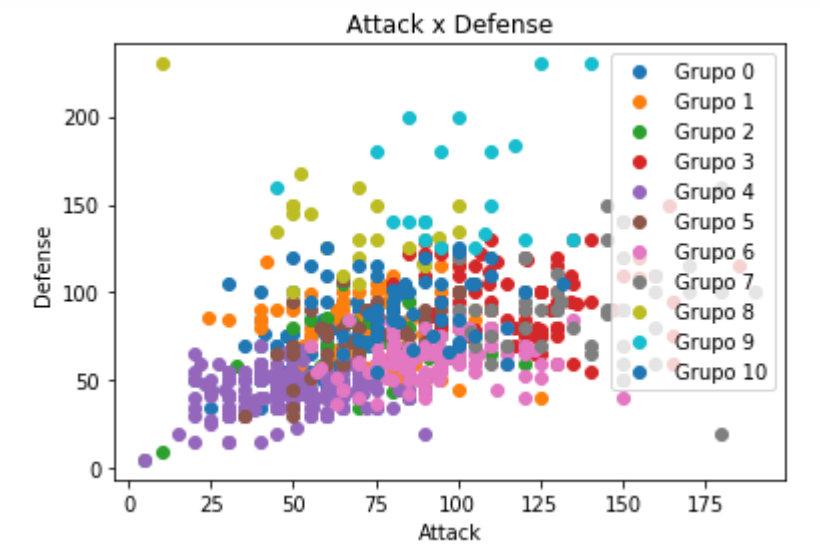
com os clusters. O problema identificado foi na “base”, em que o programa gerava saídas diferentes para um mesmo pokémon a cada vez que ele era rodado. Por exemplo, quando o código era rodado, ora Dialga tinha saída 10, ora saída 5, ora saída 2. Isso ocorria por causa de uma randomização presente na “base” que faz com que a função predict gere valores (classificações) diferentes a cada rodada. Para normalizar tal efeito, foi utilizado um código retirado do StackOverflow que mantinha as saídas originais. Logo, o programa devolvia os grupos de pokémons parecidos, baseado apenas nos respectivos status dos pokémons, sem alterar a saída de cada pokémon a cada rodada, mas ainda sem considerar os pontos fracos e fortes.

Como dito anteriormente, foram geradas 11 saídas, isto é, 11 grupos foram gerados pois este número de clusters foi considerado ideal, havendo uma boa separação dos pokémons. Por exemplo, a maioria dos pokémons iniciais se encontram no grupo 4 e, de fato, todos os iniciais se parecem. Outro exemplo, são os pokémons lendários. A maioria deles, aliás, lendários fortes ficam no grupo 7, onde se localiza a maioria dos pokémons fortes do jogo. Assim, para análise desses 11 grupos, de início, foram plotados gráficos 3D. No entanto, como em cada gráfico analisava-se apenas duas variáveis, sendo melhor fazer tal análise em um gráfico 2D. Assim, plotamos gráficos 2D, a partir de uma função com combinações dois-a-dois:

```
def plota_grafico(atributo1, atributo2):
    plt.plot(grupo0["{}".format(atributo1)], grupo0["{}".format(atributo2)], "o", label='Grupo 0')
    plt.plot(grupo1["{}".format(atributo1)], grupo1["{}".format(atributo2)], "o", label='Grupo 1')
    plt.plot(grupo2["{}".format(atributo1)], grupo2["{}".format(atributo2)], "o", label='Grupo 2')
    plt.plot(grupo3["{}".format(atributo1)], grupo3["{}".format(atributo2)], "o", label='Grupo 3')
    plt.plot(grupo4["{}".format(atributo1)], grupo4["{}".format(atributo2)], "o", label='Grupo 4')
    plt.plot(grupo5["{}".format(atributo1)], grupo5["{}".format(atributo2)], "o", label='Grupo 5')
    plt.plot(grupo6["{}".format(atributo1)], grupo6["{}".format(atributo2)], "o", label='Grupo 6')
    plt.plot(grupo7["{}".format(atributo1)], grupo7["{}".format(atributo2)], "o", label='Grupo 7')
    plt.plot(grupo8["{}".format(atributo1)], grupo8["{}".format(atributo2)], "o", label='Grupo 8')
    plt.plot(grupo9["{}".format(atributo1)], grupo9["{}".format(atributo2)], "o", label='Grupo 9')
    plt.plot(grupo10["{}".format(atributo1)], grupo10["{}".format(atributo2)], "o", label='Grupo 10')
    plt.xlabel("{}".format(atributo1))
    plt.ylabel("{}".format(atributo2))
    plt.title("{} x {}".format(atributo1,atributo2))
    plt.legend()
    plt.show()
```

Retirado do documento P3 jupyter notebook

A função recebe dois atributos, por exemplo, “Attack” e “Defense”; e ela plota um gráfico de “Attack” por “Defense”, mostrando todos os 11 grupos nesse mesmo gráfico, como mostra o gráfico a seguir:



Retirado do documento P3 jupyter notebook

Como temos que analisar 6 atributos dos status, de dois a dois, devemos analisar um total de 15 gráficos nos forneceu 15 gráficos (resultado da combinação de 6 dois a dois).

Nota-se que não há um agrupamento claro, visualmente. Isso se deve ao fato de que alguns pokémons se aproximam de outros grupos também. Mesmo que isso ocorra, analisando cada grupo em particular, nota-se que faz sentido os pokémons de um grupo em específico estarem juntos. Como já dito antes, grupos como 4 e 7 estão bem definidos, assim como os demais.

Todavia, o objetivo do nosso projeto não é observar e analisar apenas os grupos de pokémons parecidos segundo seus status. Queremos identificar o grupo de pokémon mais fortes para uma batalha contra um pokémon adversário. Logo, precisamos considerar os respectivos ponto-fracos, ponto-fortes e resistência de cada pokémon para fazer a análise.

Análise dos dados por clusterização para batalha

Na sequência, temos uma função em que os tipo 1 e tipo 2 de um pokémon qualquer, presente no dataset, são selecionados para identificar os tipos compostos de pokémon para cada um.

```
def ClassificaTipoComposto(pokemon):
    tipo1=pokedex[pokedex.Name==pokemon]['Type 1'].all()
    tipo2=pokedex[pokedex.Name==pokemon]['Type 2'].all()
    return tipo1,tipo2
```

Retirado do documento P3 jupyter notebook

Por exemplo, se utilizarmos o código “ClassificaTipoComposto(Articuno)”, o código retornará os tipos “Ice” e “Flying”.

Como temos os tipos do pokémon, é interessante analisar quais os ponto-fracos, ponto-fortes e resistência do pokémon. De início, analisamos isso apenas considerando um tipo, isto é, considerando que os pokémons possuem apenas um tipo. Por exemplo, Chimchar é do tipo fogo, apenas e, portanto, possui apenas um tipo. Assim, foi feita a seguinte função para fazer tais identificações:

```
#Define o ponto fraco de cada tipo de pokemon. REferencias de fraqueza baseadas no site: https://pokemondb.net/type
def PontoFracoPorTipo(tipo):
    if tipo=='Fairy':
        ponto_fraco=['Poison','Steel']
        ponto_forte=['Fighting','Bug','Dark']
        no_effect=['Dragon']
    elif tipo=='Dragon':
        ponto_fraco=['Ice', 'Fairy', 'Dragon']
        ponto_forte=['Fire', 'Water', 'Electric', 'Grass']
        no_effect=['None']
    elif tipo=='Ground':
        ponto_fraco=['Water','Grass','Ice',]
        ponto_forte=['Poison', 'Rock']
        no_effect=['Electric']
```

Retirado do documento P3 jupyter notebook

Note que, nesta imagem, a função não se mostra completa. Isto se deve ao fato dela ser muito grande e ocuparia muito espaço neste documento, pois existem, no total, 18 tipos; mas a mesma lógica se aplica a todos os tipos.

Assim, analisamos apenas situações em que pokémons possuem apenas um tipo. Temos que considerar pokémons que possuem dois tipos também. Essa análise é importante pois, um pokémon pode ser apenas do tipo fogo. Nesse caso, um dos seus ponto fracos seria água. No entanto, se o pokémon for do tipo fogo e água, como Volcanion, teríamos que água é ponto fraco de fogo, mas ponto-forte do tipo água.

Logo, água se torna um ataque “normal”, que não possui vantagens contra Volcanion. Para isso, foi feita a seguinte função:

```
#Dado um pokemon, retorna-se seu ponto fraco, ponto forte e resistencia
def PontosReais(pokemon):
    tipos=ClassificaTipoComposto(pokemon)
    tipo1=tipos[0]
    tipo2=tipos[1]
    first=PontoFracoPorTipo(tipo1)
    second=PontoFracoPorTipo(tipo2)
    ponto_fraco_temporario=set(first[0]+second[0])
    ponto_forte_temporario=set(first[1]+second[1])
    resistencia=set(first[2]+second[2])
    ponto_fraco_real=[]
    ponto_forte_real=[]
    for elemento in ponto_fraco_temporario:
        if elemento not in second[1]:
            if elemento not in second[2]:
                if elemento not in first[1]:
                    if elemento not in first[2]:
                        ponto_fraco_real.append(elemento)
    for elemento in ponto_forte_temporario:
        if elemento not in second[0]:
            if elemento not in second[2]:
                if elemento not in first[0]:
                    if elemento not in first[2]:
                        ponto_forte_real.append(elemento)
    return ponto_fraco_real, ponto_forte_real,resistencia
```

Retirado do documento P3 jupyter notebook

Como dito anteriormente, essa função retorna os pontos-fracos, pontos-fortes e resistência considerando também, pokémons que possuem 2 tipos. Com essas funções, já poderíamos avançar para a clusterização de pokémons considerando ponto-fracos, ponto-fortes e resistência. A função a seguir funciona de forma que, se o tipo de um pokémon for igual ao ponto fraco do pokémon adversário, seu ataque e ataque especial são dobrados. Caso um pokémon seja do mesmo tipo que o ponto forte do pokémon fornecido, seu ataque e ataque especial caem pela metade. E, ainda, se o tipo do pokémon adversário for super resistente com relação ao pokémon, seu ataque e ataque especial são zerados. Por exemplo, um pokémon do tipo Normal, não consegue atacar um do tipo Ghost, pois Ghost é “super resistente” ao tipo Normal e, portanto, os ataques de um pokémon Normal não teriam efeitos no tipo Ghost.

```python
def Adapta_df(pokemon):
    nome_do_pokemon=(pokedex[pokedex.Name==pokemon])
    Ataque=[]
    Special_ataque=[]
    if len(nome_do_pokemon)!=0:
        weakness=PontosReais(pokemon)[0]
        strength = PontosReais(pokemon)[1]
        super_resistance=PontosReais(pokemon)[2]
        for pocket_monster in nova_tabela.Name:
            tipo1,tipo2=ClassificaTipoComposto(pocket_monster)
            if tipo1 in weakness or tipo2 in weakness:
                Ataque.append(int(pokedex[pokedex['Name']==pocket_monster].Attack)*2)
                Special_ataque.append(int(pokedex[pokedex['Name']==pocket_monster]["Sp. Atk"])*2)
            elif tipo1 in strength or tipo2 in strength:
                Ataque.append(int(pokedex[pokedex['Name']==pocket_monster].Attack)*(1/2))
                Special_ataque.append(int(pokedex[pokedex['Name']==pocket_monster]["Sp. Atk"])*(1/2))
            elif tipo1 in super_resistance or tipo2 in super_resistance:
                Ataque.append(0)
                Special_ataque.append(0)
            else:
                Ataque.append(int(pokedex[pokedex['Name']==pocket_monster].Attack))
                Special_ataque.append(int(pokedex[pokedex['Name']==pocket_monster]["Sp. Atk"]))
        nova_tabela['Attack']=Ataque
        nova_tabela['Sp. Atk']=Special_ataque
        return nova_tabela
    else:
        return("Pokémon não encontrado. Verifique se a escrita está correta.")
```

Retirado do documento P3 jupyter notebook

Feita a função, fizemos uma célula no qual o usuário entra com um pokémon e o programa devolve o grupo de pokémons ideais:

```python
#Resultado final obtido
#Devolvendo o melhor grupo de pokémons para batalhar com um pokémon a escolha do leitor:
pokemon = input("Qual seu pokémon adversário? ")
tabela = Adapta_df(pokemon) #atualiza a tabela de acordo com os pontos fortes, pontos fracos e ataques sem efeitos

try:
    tabela_cluster = Cluster(tabela) #Clusteriza a tabela atualizada
    grupo_repetido=[]
    grupo_solo=[]
    ataque_max = tabela_cluster.sort_values(by="Attack")["Attack"].max() #Identifica o pokémon com maior ataque
    saida_do_pokemon=list(tabela_cluster[tabela_cluster.Attack==ataque_max].saida)#Identifica a qual cluster o pokémon forte pert
    if len(saida_do_pokemon)!=0:
        for element in tabela_cluster.saida:
            if element == int(saida_do_pokemon[0]):
                nome_real = tabela_cluster[tabela_cluster["saida"]==element].Name
                grupo_repetido.append(nome_real)
        for p in grupo_repetido[0]:
            grupo_solo.append(p)
        print("Os melhores pokémons contra {} são: {}".format(pokemon, grupo_solo))
    else:
        print("Esse pokémon não existe ou o nome foi digitado de forma incorreta ou ainda ele foi reservado para a parte teste.")
except AttributeError:
    print("Esse pokémon não existe ou o nome foi digitado de forma incorreta ou ainda ele foi reservado para a parte teste.")
```

Retirado do documento P3 jupyter notebook

Transformamos a “saída_do_pokemon” em uma lista pois quando pegávamos o ataque máximo, havia vários ataques iguais e o programa dava erro. Logo, já que são valores iguais, bastou apenas pegar um dos valores da lista e utilizá-lo para pegar o grupo equivalente ao pokémon de maior ataque.

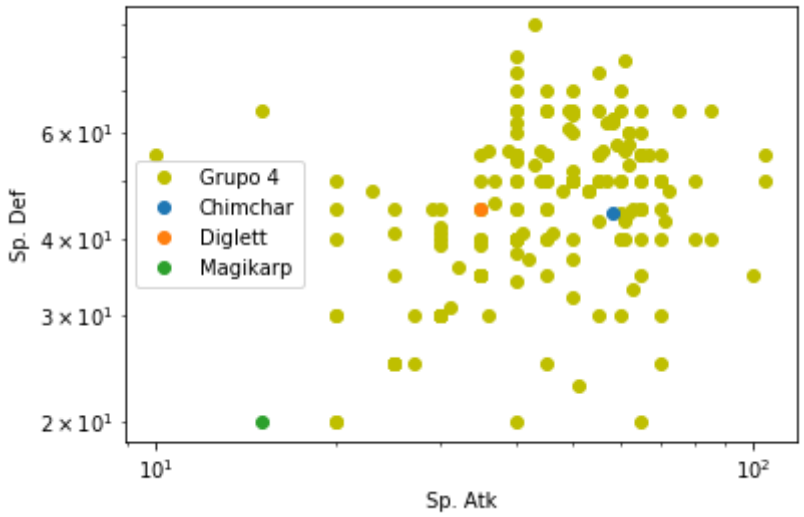
Dessa forma conseguimos responder a pergunta dada no início. Porém, temos que testar o grau de confiança do programa. Para isso, utilizamos os grupo dos 10

pokémons separados e colocamos na função para ver em qual grupo eles eram colocados. Em seguida foi feita uma função para plotar o grupo em que os pokémons foram colocados, destacando os pokémons do grupo separado, tal como a imagem abaixo:

```
def comparacao(lista, at1,at2,s):
    plt.plot(pokedex[pokedex.saida==s][at1],pokedex[pokedex.saida==s][at2],"yo", label="Grupo {}".format(s))
    for pokemon in lista:
        plt.plot(teste[teste.Name==pokemon][at1],teste[teste.Name==pokemon][at2],"o", label="{}".format(pokemon))
    plt.xlabel('{}'.format(at1))
    plt.ylabel('{}'.format(at2))
    plt.loglog()
    plt.legend()
    plt.show()
```

Retirado do documento P3 jupyter notebook

Para utilizar a função, colocamos a lista dos pokémons de teste do mesmo grupo, os atributos que queremos comparar e o grupo em que eles foram classificados. Dados os argumentos, a função devolve um gráfico no estilo:



Retirado do documento P3 jupyter notebook

Plotados os gráficos, foi analisado se o grupo em que os pokémons foram colocados era coerente. Dos 10 pokémons, apenas um não era coerente perante sua classificação, que foi o Magikarp. Como podemos ver no gráfico acima, Magikarp está muito isolado tanto em defesa especial quanto em ataque especial, reforçando a ideia de que não pertence ao grupo 4. Logo, Magikarp é um outlier do grupo 4.

Conclusão

Conclui-se, então, que o classificador via cluster do nosso projeto é razoavelmente bom, pois ele possui um acerto de 90% (considerando que foram testados apenas 10 pokémons de 800). Com base nessa acurácia, podemos dizer que o usuário teria confiança de utilizar nosso programa para sua decisão em um batalha. Porém, não afirmamos que o nosso classificador está perfeito, pois ainda podemos melhorá-lo, como por exemplo, classificar corretamente o Magikarp. Justamente por se tratar de um cluster, nós que adotamos um grau de semelhança (demonstrado pelo número de grupos criados pelo cluster). Isto é, é o grupo que analisa se Pikachu e Pichu, por exemplo são parecidos. Logo, há espaço para questionamento sobre certos pokémons classificados.

Podemos melhorar o classificador, aumentando o dataset de análise incluindo, por exemplo, pokémons da região de Alola. Podemos testar outros números de clusters com ou sem randomização no predict (este é um ponto que foi levantado na célula de código da primeira clusterização, em que a randomização pode alterar os pokémons de alguns grupos, devido a saída diferente gerada pelo predict). Vale ressaltar que o grau de semelhança ou se um pokémon se parece com outro ou ainda se faz sentido um pokémon pertencer a um grupo, varia de pessoa para pessoa. Assim, mesmo que tenhamos encontrado um número ideal de 11 clusters, outras pessoas poderiam ter outras opiniões com número de clusters diferentes. No entanto, se adotarmos um número de clusters muito alto, a classificação acaba ficando distorcida e muito específica, podendo até separar pokémons que são de fato parecidos, devido a um status levemente diferente, podendo devolver poucas opções de pokémons e talvez desconsiderar um que seria ideal para a batalha. Por outro lado, se adotarmos um número de clusters muito baixo, o classificador junta pokémons parecidos junto com alguns que têm atributos bem distintos, sendo muito pouco específico, devolvendo pokémons que não seriam ideais para a batalha. Assim, o ideal seria mexer no número sem extrapolar muito.

De forma geral, nosso programa funciona e devolve um grupo de pokémons fortes condizentes com o pokémon adversário.

Contribuição de cada membro

Maurício: Contribuiu para a análise do boxplot e fez a introdução às funções de clusterização. Com a ajuda de Theo, fez a maior parte da função que define o ponto fraco por tipo de pokémon. Criou, também, as funções de classificar o ponto fraco, ponto forte e resistência de cada pokémon, que foi usado para responder à pergunta inicial. Ajudou Fernando a fazer a função “Adapta_df” com pequenos detalhes. Finalmente, fez a análise de validação da clusterização: analisou os pokémons teste com suas respectivas classificações de saída com os grupos. Contribuição significativa no relatório.

Fernando: Havia feito seaborn para analisar a correlação entre os status dos pokémons, mas optamos por não utilizar no final. Ajudou na criação de algumas funções para deixar o programa mais limpo e mais fácil de utilizar. Por fim, elaborou a conclusão baseada na análise da clusterização. Contribuição significativa no relatório.

Theo: Plotou gráficos 3D com cluster de variáveis duas-a-duas, que depois por decisão conjunta do grupo, foram retirados e substituídas por gráficos 2D. Verificou e corrigiu alguns dos pontos fracos, fortes e neutros de cada Pokémon na terceira função. Antes da conclusão, plotou gráficos 2D de comparação de Pokémons, a partir de uma função, com a ajuda de Maurício, de grupos do cluster de variáveis duas-a-duas. Contribuição significativa no relatório.

Referências:

"clustering_SKL.ipynb" da aula 27
https://stackoverflow.com/questions/25921762/changes-of-clustering-results-after-each-time-run-in-python-scikit-learn
https://www.kaggle.com/abcsds/pokemon