

# Estudo Metódico da Entropia da Informação

Aplicada a problemas computacionais

# Fórmulas

- Entropia de Shanon

$$H(X) = - \sum_{i=1}^n p(x_i) * \log_b(p(x_i))$$

- Probabilidades

$$\left( \sum_{i=1}^n p(x_i) = \frac{x_i}{n} \right) = 1$$

- Normalização





$$H_N = \frac{H(X)}{\log_b(n)}$$

$$H_N = \frac{H(X) * \log_e(2)}{\log_e(n)} \rightarrow \text{Normalized specific entropy}$$

$$H_N = \frac{H(X) * N * \log_e(2)}{\log_e(n)} \rightarrow \text{Total normalized extensive entropy}$$

Algumas  
ferramentas  
usam logb  
como logn ou  
log2

# Quais as questões/problema

- ◉ Calcular a entropia
  - ◉ Para dados discretos 
  - ◉ Para dados contínuos 
- ◉ Atualizar a entropia
  - ◉ Em *batch* 
  - ◉ On stream 
- ◉ Usar a entropia para melhorar o ensemble
  - ◉ Aliado a *Feature Importance* resultando em um vetor de pesos para ponderar as decisões das árvores (antes do voto majoritário)
  - ◉ Monitorar a entropia como alternativa a detecção do *concept drift*

# Analizando a estrutura da entropia

$$H(X) = - \sum_{i=1}^n p(x_i) * \log_e(p(x_i))$$

- Usando uma propriedade de logaritmo, podemos mudar essa equação de  $H(X)$ ,

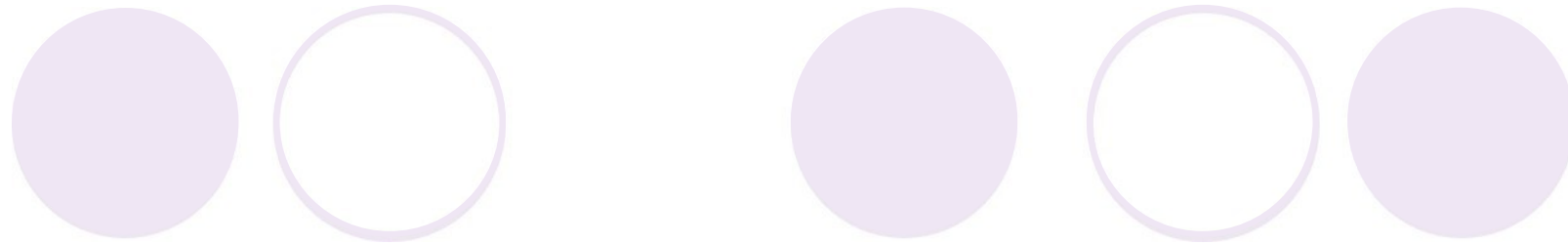
$$-\log_e a = \frac{1}{a}$$

Então,

$$H(X) = \sum_{i=1}^n p(x_i) * \log_e\left(\frac{1}{p(x_i)}\right)$$

# ○ que sabemos da entropia

- ◉ É uma medida da incerteza em uma variável aleatória
- ◉ Surgiu na Teoria da Informação e funciona bem para dados discretos, porém necessita de adaptações para dados contínuos
- ◉ As ferramentas disponíveis hoje Scipy, ScikitLearn e River não aprofundam no trabalho da entropia, chegando a adotar a primeira forma de sua equação (com sinal negativo)
- ◉ As probabilidades usada na entropia são uma distribuição normal
- ◉ As probabilidades usada na entropia são calculadas a partir da frequência de cada elemento único
- ◉ Algumas fórmulas “sugerem” o uso de logaritmo na base dois, porém as ferramentas de referência usam logaritmo natural



- ◉ Uma definição equivalente de entropia é o valor esperado da auto-informação de uma variável.
- ◉ base 2 fornece a unidade de bits (ou "shannons"), enquanto a base  $e$  fornece "unidades naturais" nat e a base 10 fornece unidades de "dits", "bans" ou "hartleys".
- ◉ A ideia central da teoria da informação é que o "valor informacional" de uma mensagem comunicada depende do grau em que o conteúdo da mensagem é surpreendente. Se ocorrer um evento altamente provável, a mensagem carrega muito pouca informação. Por outro lado, se ocorrer um evento altamente improvável, a mensagem é muito mais informativa.

# Cont...

- ◉ Quantifica o valor esperado da informação contida em uma mensagem.
- ◉ Proposta por Claude E. Shannon em 1948.
- ◉ Existem outras entropias: Entropy MillerMadow; Entropy Dirichlet; Entropy ChaoShen; Entropy NSB; Entropy Shrink
- ◉ O sinal de menos é utilizado porque os valores menores que o logaritmo 1 são negativos. Lembrando que a distribuição de probabilidades é uma distribuição normal então naturalmente cada  $x_i$  é menor ou igual a 1.
- ◉ A expressão,

$$\log_e\left(\frac{1}{p(x_i)}\right)$$

Também é chamada de incerteza ou surpresa, quanto mais baixa a probabilidade  $p(x_i)$ , maior a incerteza ou a surpresa

# Entendendo tudo...

- ◉  $\log_n$  na fórmula de Shannon provoca uma “penalização” para as probabilidades maiores.
- ◉ Isto está certo? Sim, porque a entropia mede a incerteza, então quanto maior a incerteza (probabilidade baixa) maior será seu peso atribuído por  $\log_n$ .

|   | A              | B   | C | D                                  | E   |
|---|----------------|-----|---|------------------------------------|---|
| 1 | Probabilidades |     |   | <b><math>\log_n(1/p(x))</math></b> | <b><math>p(x_i) \cdot \log_n(1/p(x))</math></b> |
| 2 | $p(x_1)$       | 0,9 |   | 0,105360516                        | 0,094824464                                     |
| 3 | $p(x_2)$       | 0,1 |   | 2,302585093                        | 0,230258509                                     |
| 4 |                |     |   |                                    |   |
| 5 |                |     |   | <b>Entropia</b>                    | 0,325082973                                     |
| 6 |                |     |   |                                    |   |



# Alterando os conceitos

- ◉ Dado que a entropia **mede a incerteza** “penalizando” as probabilidades altas e “maximizando” as probabilidades baixas, nós podemos modificar essa fórmula ligeiramente para uma “**medida de certeza**”. Como? Usando a função inversa de  $\log_n$  (uma função exponencial).
- ◉ A função inversa do logaritmo natural é,
$$f^{-1}(x) = e^x$$
- ◉ E a função inversa do logaritmo base 2 é,
$$f^{-1}(x) = 2^x$$
- ◉ Ou também pode se calcular usando a entropia da incerteza normalizada,

$$f^{-1}(x) = 1 - \frac{H(X)}{\log_2(n)}$$

# Vejamos...

- ◉ Calculamos a entropia a partir de uma lista de probabilidades.
- ◉ Essa lista é uma distribuição normal, ou seja, somando todas as probabilidades teremos o resultado 1.
- ◉ Essa distribuição pode ter  $n$  elementos (no exemplo anterior 2 foram usados apenas para ilustração)
- ◉ A entropia é a soma dos resultados calculando a probabilidade de  $X_i$  vezes o logaritmo natural de 1 dividido por  $X_i$ . Em resumo, essa soma não significa muita coisa e pode resultar em valores difíceis de manipular. Dado isso é necessário sua regularização para uma distribuição mais interessante de se manipular.
- ◉ É importante que essa “regularização” resulte em uma distribuição normal.
- ◉ A dificuldade de manipular a entropia é que os resultados dos cálculos nem sempre vão estar dentro de faixas de valores conhecidos e não... não.. a entropia não é uma distribuição normal, normalizar/regularizar ela permite colocar ela numa faixa de valores conhecida e fácil de manipular.

# Falando das probabilidades

- Em tese podemos usar a entropia em qualquer situação que tenhamos as probabilidades conhecidas.
- Em vias práticas (se tratando de valores discretos) calculamos as probabilidades contando a frequência de valores únicos.

$X = [1, 2, 2, 3, 1, 4, 5, 4, 5, 5, 3, 2, 2, 1, 2]$

Valores únicos:  $[1, 2, 3, 4, 5]$

| Valor Único | Frequência | Probabilidade |
|-------------|------------|---------------|
| 1           | 3          | $3/15=0,2$    |
| 2           | 5          | $5/15=0,33$   |
| 3           | 2          | $2/15=0,13$   |
| 4           | 2          | $2/15=0,13$   |
| 5           | 3          | $3/15=0,2$    |

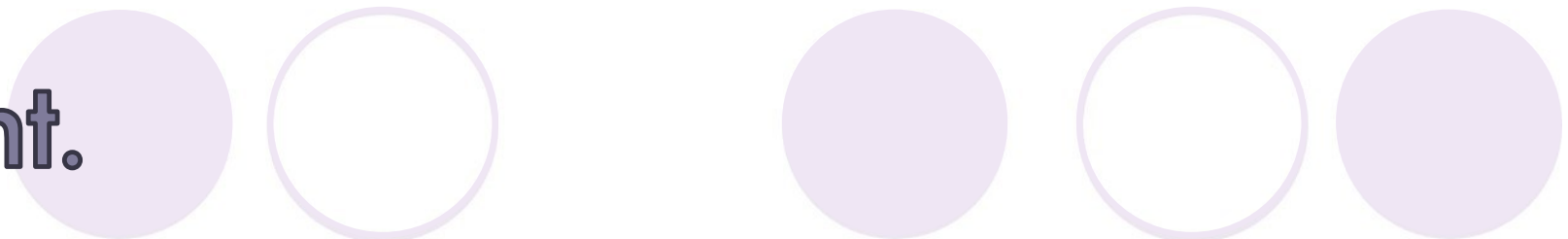
# Corrigindo o erro

- ◉ Dado que a soma das probabilidades é uma distribuição normal, podemos calcular o erro nesta etapa usando a fórmula,

$$E(X) = 1 - \sum_{i=1}^n p(x_i) = 1 - \sum_{i=1}^n \frac{x_i}{n}$$

O erro resulta da imprecisão dos cálculos numéricos, seja por truncamento ou limitação da representação de valores contínuos.

Cont.

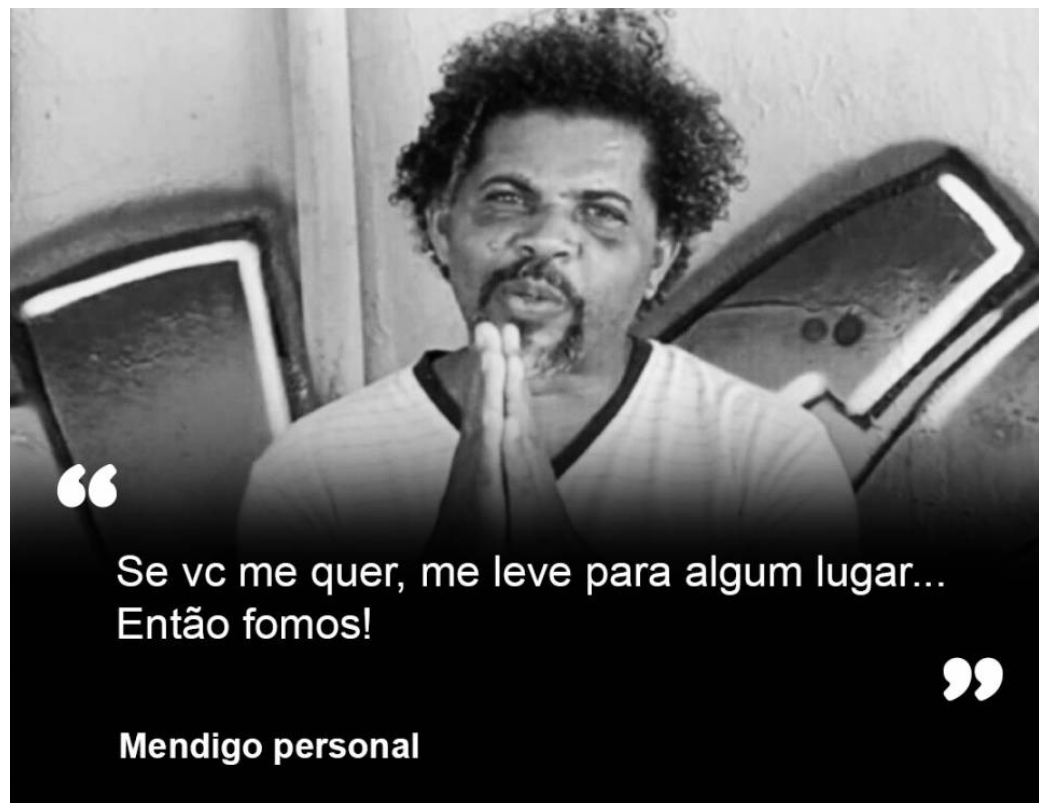


- ◉ Levando o erro em consideração, podemos aplicar ele na fórmula da entropia, tornando-a ligeiramente mais precisa.

$$H(X)' = \left( \sum_{i=1}^n p(x_i) * \log_e \left( \frac{1}{p(x_i)} \right) \right) + \left( \left( 1 - \sum_{i=1}^n p(x_i) \right) * \log_e \left( \frac{1}{1 - \sum_{i=1}^n p(x_i)} \right) \right)$$

# Pausa para os memes

O que a entropia disse para as probabilidades?



# Daqui pra frente começa os problemas

- ◉ O caso dos valores contínuos
- ◉ Atualizando a entropia no fluxo



# ○ caso dos valores contínuos

- A entropia é independente do “sistema de valores”, podemos afirmar isso porque para calcular a entropia dependemos das probabilidades.
- Para valores discretos calculamos as probabilidades a partir da frequência dos valores únicos.
- Porém para valores contínuos não conseguimos contar essa frequência. Por que?
  - Os valores contínuos são,  
$$x \in \mathbb{R} \rightarrow [-\infty, +\infty]$$



# Cont...

- ◉ A questão principal é como calcular a probabilidade para números reais?
- ◉ Indo mais longe... Como fazer isso em fluxo?
- ◉ São pequenas questões que podem mudar claramente a interpretação da entropia.
- ◉ Uma estratégia seria discretizar esses valores, mas vejamos,
  - ◉ Num contexto onde você conhece a distribuição dos valores do conjunto você consegue discretizar e ter uma distribuição interessante (ou equitativa)
  - ◉ Mas e em fluxo? Quando você não conhece os valores que irão chegar, somente conhecendo eles quando eles estão na porta?
- ◉ Outra estratégia é a entropia diferencial

# Cont...

- ◉ Discretizar → Uma transformação de discretização mapeará variáveis numéricas em valores discretos.
- ◉ Chegamos a dois pontos,
  - ◉ Ignorar as mudanças do fluxo e “pré-definir” a partir dos dados de treinamento uma maneira de discretizar os dados.
  - ◉ Ter uma estrutura flexível que ajuste essa “discretização” ao longo do tempo.

# Falando de discretização

- ◉ Suponhamos o peso de uma balança que esteja entre 10,6 quilos de carne e 11,5. Nos Números Reais entre 10,6 e 11,5 temos infinitos números. Por conta disso convençionamos que 11 representa estes valores infinitos nesta intervalo. Porque 11? Usamos regras de arredondamento que todo valor acima de 0,6 ou menor que 0,5 é arredondando para o inteiro positivo mais próximo.
- ◉ Quando falamos em discretização de números reais “intuitivamente” estamos construindo uma noção de agrupamento de valores.

# Usando algo de python

```
Import numpy as np
```

```
size_attr = 12
```

```
n_partitions = 4
```

```
attr_a_floats = np.random.uniform(0.5, 3.0, size=(1, size_attr))[0]
```

```
# linspace -> Retorna números com espaçamento uniforme em um intervalo especificado
```

```
n_bins = np.linspace(np.amin(attr_a_floats), np.amax(attr_a_floats),  
n_partitions, dtype=np.float64)
```

```
# digitize -> Retorna os índices dos compartimentos aos quais cada valor na matriz de entrada pertence.
```

```
discretized_attr_a_floats = np.digitize(attr_a_floats, n_bins)
```

```
print('N bins: ', n_bins)
```

```
for a, d in zip(attr_a_floats, discretized_attr_a_floats):
```

```
    print('Attr value: ', a, " \tDiscretized: ", d)
```

Cont.

N bins: [0.51143542 1.32465918 2.13788293 2.95110668]

Attr value: 2.419490061591645 Discretized: 3

Attr value: 0.5114354236377701 Discretized: 1

Attr value: 0.7974285461997217 Discretized: 1

Attr value: 2.7067111004255997 Discretized: 3

Attr value: 1.1326619774019158 Discretized: 1

Attr value: 1.4759456004483604 Discretized: 2

Attr value: 1.3692895089661914 Discretized: 2

Attr value: 2.234159070749879 Discretized: 3

Attr value: 2.201222581311886 Discretized: 3

Attr value: 2.951106684404706 Discretized: 4

Attr value: 2.7675886761072532 Discretized: 3

Attr value: 1.761250225325651 Discretized: 2

# Outra bibliotecas pra fazer isso

- ◉ Sklearn
- ◉ Orange3
- ◉ Pandas

# Discretização

- Os valores da variável são agrupados em compartimentos discretos e a cada compartimento é atribuído um número inteiro exclusivo, de modo que o relacionamento ordinal entre os compartimentos seja preservado.
- No modelo “tradicional” aprende-se no treinamento e mantém-se fixo nos testes.
- Diferentes métodos para agrupar os valores em  $k$  compartimentos discretos podem ser usados; técnicas comuns incluem
  - Uniforme:** Cada bin tem a mesma largura no intervalo de valores possíveis para a variável.
  - Quantil:** Cada bin tem o mesmo número de valores, divididos com base em percentis.
  - Clustered:** Os clusters são identificados e os exemplos são atribuídos a cada grupo.
  - Target:** Baseada nas amostras ou todos os dados em um conjunto de dados.

# Discretização e Entropia

- ◉ Uma vez que conseguimos discretizar os valores contínuos, podemos estimar as probabilidades e então calcular a entropia.
- ◉ Aqui entra um ponto: Se o mecanismo de “discretização” não for flexível (ou seja, adaptar-se ao fluxo) vai ocorrer que determinado “n\_partition (k\_bin)” receba uma maior contagem de elementos, e portanto uma maior probabilidade. Sai-se de uma distribuição probabilística normal para uma distribuição probabilística uniforme.
- ◉ Como demonstrado elementos com maior probabilidade tendem a ser “suprimidos” pela no cálculo da entropia.



# Atualizando a entropia no fluxo

Vamos supor o seguinte vetor  $X = [5, 5, 5, 3, 2, 1, 5, 1, 5, 3]$  e nós temos a seguinte sequência deste vetor que será acrescentada incrementalmente  $[4, 4, 1, 2, 3]$ . Nós também vamos fixar um limite de 10 elementos no tamanho de  $X$  (threshold), assim, acrescentemos um elemento ao final do vetor  $X$  e removemos o primeiro.

| X                     | Entropia | Ent. Norm; |
|-----------------------|----------|------------|
| [5,5,5,3,2,1,5,1,5,3] | 1.2206   | 0.3674     |
| [5,5,3,2,1,5,1,5,3,4] | 1.4708   | 0.4427     |
| [5,3,2,1,5,1,5,3,4,4] | 1.5571   | 0.4687     |
| [3,2,1,5,1,5,3,4,4,1] | 1.5571   | 0.4687     |
| [2,1,5,1,5,3,4,4,1,2] | 1.5571   | 0.4687     |
| [1,5,1,5,3,4,4,1,2,3] | 1.5571   | 0.4687     |

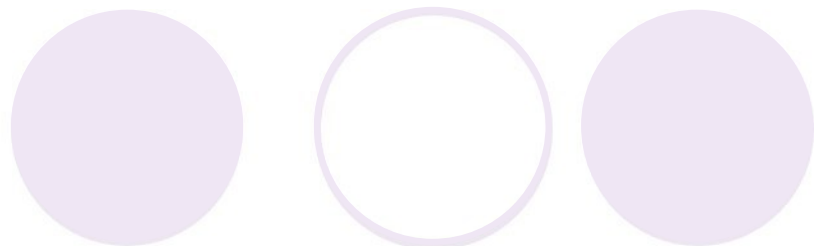


# Cont...

A entropia aqui foi calculada usando o River

Por que a entropia permanece igual mesmo mudando os dados?

Porque a distribuição probabilística se mante a mesma.



| X                     | Entropia | Ent. Norm; |
|-----------------------|----------|------------|
| [5,5,5,3,2,1,5,1,5,3] | 1.2206   | 0.3674     |
| [5,5,3,2,1,5,1,5,3,4] | 1.4708   | 0.4427     |
| [5,3,2,1,5,1,5,3,4,4] | 1.5571   | 0.4687     |
| [3,2,1,5,1,5,3,4,4,1] | 1.5571   | 0.4687     |
| [2,1,5,1,5,3,4,4,1,2] | 1.5571   | 0.4687     |
| [1,5,1,5,3,4,4,1,2,3] | 1.5571   | 0.4687     |

# Cont...

Fato adicional: Não dá pra calcular a entropia atualizada adicionando ou removendo entropia, a justificava está simplesmente na distribuição de probabilidades, mas vejamos um exemplo,

1. Iniciamos  $X = [5, 5, 5, 3, 2, 1, 5, 1, 5, 3]$ ,  
a entropia é 1.2206
2. Adicionamos +1 elemento  $X = [5, 5, 5, 3, 2, 1, 5, 1, 5, 3, 4]$ ,  
A entropia é 1.4142
- 3.1. Subtraímos a entropia atualizada da inicial, temos  
 $1.4142 - 1.2206 = 0.1936$
- 3.2. Subtraímos a diferença da entropia atualizada,  
 $1.4142 - (1.4142 - 1.2206) = 1.2206$
- 3.3 Adicionamos a entropia inicial a atual,  
 $1.2206 + 1.4142 = 2,6348$
- 3.4 Fazemos a média,  
 $(1.2206 + 1.4142/2) = 1.3174$
- 3.4 Adicionamos a diferença da entropia na entropia inicial,  
 $1.2206 + (1.4142 - 1.2206) = 1,4142$

# Cont..

Mantendo os parâmetros do exemplo anterior, nenhum das situações (3.x) do algoritmo ao lado consegue chegar na entropia correta que é 1.4708 para o vetor [5,5,3,2,1,5,1,5,3,4].

Claramente o jeito correto é fazer a rotação do vetor (adiciona no final e remove o primeiro). Mas aqui está sendo usado a lista de elementos.

1. Iniciamos  $X = [5,5,5,3,2,1,5,1,5,3]$ ,  
a entropia é 1.2206
2. Adicionamos +1 elemento  $X = [5,5,5,3,2,1,5,1,5,3,4]$ ,  
A entropia é 1.4142
- 3.1. Subtraímos a entropia atualizada da inicial, temos  
 $1.4142 - 1.2206 = 0.1936$
- 3.2. Subtraímos a diferença da entropia atualizada,  
 $1.4142 - (1.4142 - 1.2206) = 1.2206$
- 3.3 Adicionamos a entropia inicial a atual,  
 $1.2206 + 1.4142 = 2.6348$
- 3.4 Fazemos a média,  
 $(1.2206 + 1.4142/2) = 1.3174$
- 3.4 Adicionamos a diferença da entropia na entropia inicial,  
 $1.2206 + (1.4142 - 1.2206) = 1.4142$

# Cont...

- ◉ Vejamos um exemplo mantendo um vetor de entropia, para  $X = [5, 5, 5, 3, 2, 1, 5, 1, 5, 3]$  e adicionando  $[4]$ .
- ◉ Tente repetir o algoritmo anterior e veja que também não vai dar certo.
- ◉ Motivo: O resultado da entropia depende unicamente da distribuição de probabilidades.
- ◉ Solução: Guardar um vetor não com as entropias calculadas e sim com a frequência dos valores discretos.

| X                     | Entropia           |
|-----------------------|--------------------|
| [5]                   | 0                  |
| [5,5]                 | 0                  |
| [5,5,5]               | 0                  |
| [5,5,5,3]             | 0.5623350933205757 |
| [5,5,5,3,2]           | 0.9502704577053684 |
| [5,5,5,3,2,1]         | 1.242453223487698  |
| [5,5,5,3,2,1,5]       | 1.1537418565661257 |
| [5,5,5,3,2,1,5,1]     | 1.2130074896216003 |
| [5,5,5,3,2,1,5,1,5]   | 1.149059629674195  |
| [5,5,5,3,2,1,5,1,5,3] | 1.22060720293807   |
| [5,5,3,2,1,5,1,5,3,4] | 1.4708083956432574 |

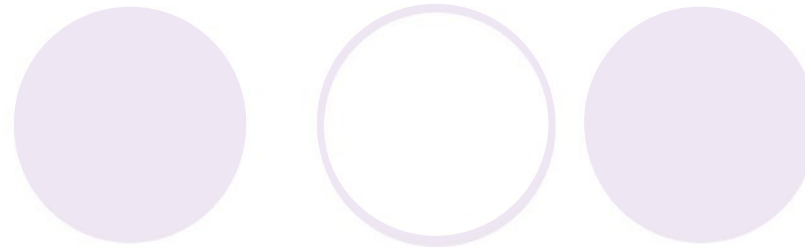
# Por que atualizar a entropia no fluxo?

- ◉ Nos baseamos em um vetor com a frequência de cada elemento único. Vamos imaginar que um atributo X assuma 4 valores discretos ao longo do fluxo:
  - ◉ Se as probabilidades desses 4 valores forem iguais então a entropia normalizada será próximo a 0,5.
  - ◉ Se as probabilidades existirem somente para um valor então a entropia será igual 0.
- ◉ Nós podemos até saber o range de um atributo, mas não sabemos quando ou como isto chegará no fluxo e isso é a evidência da necessidade de um mecanismo dinâmico e flexível para manter a entropia atualizada.
  - ◉ Se decidimos calcular a entropia considerando todos os possíveis valores discretos de um atributo teremos os efeitos contrários na interpretação da entropia. Grandes possibilidade de erro de divisão por zero.

# Relação entre entropia e pré-processamento

- ◉ Qual nossa definição de fluxo de dados?
  - ◉ Vamos ver vários casos de stream em que ajustamos o pré-processamento
- ◉ Orientações gerais
  - ◉ O mesmo conjunto de dados é usado (stream sintético)
  - ◉ O mesmo classificador é utilizado (uma DT extraída da RF)
  - ◉ Alteramos apenas o pré-processamento
  - ◉ Escrevendo camadas de adaptação onde é necessário
  - ◉ Avaliamos a acurácia dos dados de teste e a entropia resultante

# Código python



```
pip install git+https://github.com/online-ml/river --upgrade
```

```
import pandas as pd
```

```
import numpy as np
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import accuracy_score
```

```
from sklearn.preprocessing import StandardScaler
```

```
from river import preprocessing
```

```
from river import synth
```

```
from river import stream
```

```
def p_entropy(vetor):
```

```
    unique, counts = np.unique(vetor, return_counts=True)
```

```
    sum_occurrence = sum(counts)
```

```
    probabilitys = [SOi/sum_occurrence for SOi in counts]
```

```
    hX = [proba * np.log(1/proba) for proba in probabilitys]
```

```
    HX = sum(hX)
```

```
    Hxn = HX / np.log(len(vetor))
```

```
    return (HX, Hxn)
```



# Código python

```
primary_stream = synth.Agrawal(classification_function=0, seed=100, balance_classes=False,  
                                perturbation=0.1)
```

```
second_stream = synth.Agrawal(classification_function=2, seed=200, balance_classes=False,  
                                perturbation=0.2)
```

```
dataset = synth.ConceptDriftStream(stream=primary_stream, drift_stream=second_stream,  
                                    position=800, width=500, seed=110, alpha=None)
```

```
all_stream = [np.append(list(X.values()), y) for X, y in dataset.take(10000)]
```

```
xt, yt = dataset.take(2)
```

```
names_features = list(xt[0].keys())
```

```
names_features = names_features.append('target')
```

```
xt, yt = dataset.take(2)
```

```
names_features = list(xt[0].keys())
```

```
names_features.append('target')
```

```
df_stream = pd.DataFrame(all_stream, columns=names_features)
```

# Código python

```
X = df_stream[df_stream.columns[:-1]]
y = df_stream['target']

X_train = X[:1000]
y_train = y[:1000]
X_test = X[1000:]
y_test = y[1000:]

col_names = X.columns

# Sem escalonamento
clf_rf = RandomForestClassifier(n_estimators=100, criterion='entropy', max_features='auto', bootstrap=True,
                               oob_score=True, random_state=210, warm_start=True, n_jobs=-1
)
clf_rf = clf_rf.fit(X_train.to_numpy(), y_train.to_numpy())

dt0 = clf_rf.estimators_[0]

y_pred = dt0.predict(X_test.to_numpy())
```

# Código python

```
acc = accuracy_score(y_test.to_numpy(), y_pred)
print('Acc: ', acc)
```

```
ent_train = {}
ent_test = {}
for i, cl in enumerate(col_names):
    ent_train[str(cl)] = p_entropy(X_train[cl].to_numpy())
    ent_test[str(cl)] = p_entropy(X_test[cl].to_numpy())
```

```
# Com escalonamento de parâmetros fixo
skl_scaler = StandardScaler()
X_train = skl_scaler.fit_transform(X_train)
```

```
clf_rf = RandomForestClassifier(n_estimators=100, criterion='entropy', max_features='auto', bootstrap=True,
                               oob_score=True, random_state=210, warm_start=True, n_jobs=-1)
clf_rf = clf_rf.fit(X_train, y_train)
```

```
dt0 = clf_rf.estimators_[0]
X_test = skl_scaler.transform(X_test)
```

# Código python

```
y_pred = dt0.predict(X_test)
```

```
acc = accuracy_score(y_test, y_pred)  
print('Acc: ', acc)
```

```
ent_train = {}  
ent_test = {}  
for i, cl in enumerate(col_names):  
    ent_train[str(cl)] = p_entropy(X_train[:, i])  
    ent_test[str(cl)] = p_entropy(X_test[:, i])
```

```
print(ent_train)  
print('-'*5)  
print(ent_test)
```

```
# Com escalonamento dinâmico 1
```

```
rvl_scaler = preprocessing.StandardScaler()  
X_train = rvl_scaler.learn_many(X_train).transform_many(X_train)
```

```
clf_rf = RandomForestClassifier(n_estimators=100, criterion='entropy', max_features='auto', bootstrap=True, oob_score=True,  
                               random_state=210, warm_start=True, n_jobs=-1)  
clf_rf = clf_rf.fit(X_train.to_numpy(), y_train.to_numpy())
```

# Código python

```
dt0 = clf_rf.estimators_[0]  
X_test = rvl_scaler.learn_many(X_test).transform_many(X_test)
```

```
y_pred = dt0.predict(X_test.to_numpy())
```

```
acc = accuracy_score(y_test, y_pred)  
print('Acc: ', acc)
```

```
ent_train = {}  
ent_test = {}  
for i, cl in enumerate(col_names):  
    ent_train[str(cl)] = p_entropy(X_train[cl])  
    ent_test[str(cl)] = p_entropy(X_test[cl])
```

```
print(ent_train)  
print('-'*5)  
print(ent_test)
```

```
# Com escalonamento dinâmico 2
```

```
rvl_scaler = preprocessing.StandardScaler()
```

```
X_train = [rvl_scaler.learn_one(dict(zip(col_names, Xi))).transform_one(dict(zip(col_names, Xi))) for Xi in X_train.values]
```

```
X_train = pd.DataFrame(X_train)
```

# Código python

```
clf_rf = RandomForestClassifier(n_estimators=100,criterion='entropy',max_features='auto', bootstrap=True,  
                               oob_score=True,random_state=210, warm_start=True,n_jobs=-1)  
clf_rf = clf_rf.fit(X_train.to_numpy(), y_train.to_numpy())  
  
dt0 = clf_rf.estimators_[0]  
X_test = [rvl_scaler.learn_one(dict(zip(col_names, Xi))).transform_one(dict(zip(col_names, Xi))) for Xi in X_test.values]  
X_test = pd.DataFrame(X_test)  
  
y_pred = dt0.predict(X_test.to_numpy())  
  
acc = accuracy_score(y_test, y_pred)  
print('Acc: ', acc)  
ent_train = {}  
ent_test = {}  
for i, cl in enumerate(col_names):  
    ent_train[str(cl)] = p_entropy(X_train[cl])  
    ent_test[str(cl)] = p_entropy(X_test[cl])  
  
print(ent_train)  
print('-'*5)  
print(ent_test)
```

# Resultados da Acurácia por Estratégia

| Estratégia/Pré-processamento     | Acurácia           |
|----------------------------------|--------------------|
| Sem escalonamento                | 0.5423333333333333 |
| Escalonamento de parâmetros fixo | 0.542              |
| Escalonamento dinâmico 1         | 0.5436666666666666 |
| Escalonamento dinâmico 2         | 0.554              |

# Resultados da Entropia por estratégia nos dados de treinamento (1k)

|                       | Atributos/Entropia ( $\pm 0,001$ )             |            |       |        |       |         |        |        |       |
|-----------------------|--|------------|-------|--------|-------|---------|--------|--------|-------|
| Estratégia            | salary   | commission | age   | elevel | car   | zipcode | hvalue | hyears | loan  |
| Sem escalonamento     | 6.663  | 3.141      | 4.049 | 1.608  | 2.984 | 2.192   | 1.792  | 3.367  | 6.667 |
| Escal. de param. fixo | 6.663  | 3.141      | 4.049 | 1.608  | 2.984 | 2.192   | 1.792  | 3.367  | 6.667 |
| Escal. Dinâmico 1     | 6.663  | 3.141      | 4.049 | 1.608  | 2.984 | 2.192   | 1.792  | 3.367  | 6.667 |
| Escal. Dinâmico 2     | 6.907  | 6.906      | 6.907 | 6.906  | 6.907 | 6.904   | 6.906  | 6.907  | 6.907 |
|                       | Atributos/Entropia Normalizada ( $\pm 0,001$ ) |            |       |        |       |         |        |        |       |
| Estratégia            | salary   | commission | age   | elevel | car   | zipcode | hvalue | hyears | loan  |
| Sem escalonamento     | 0.964  | 0.454      | 0.586 | 0.232  | 0.432 | 0.317   | 0.259  | 0.487  | 0.965 |
| Escal. de param. fixo | 0.964  | 0.454      | 0.586 | 0.232  | 0.432 | 0.317   | 0.259  | 0.487  | 0.965 |
| Escal. Dinâmico 1     | 0.964  | 0.454      | 0.586 | 0.232  | 0.432 | 0.317   | 0.259  | 0.487  | 0.965 |
| Escal. Dinâmico 2     | 1.0  | 0.99       | 1.0   | 0.999  | 1.0   | 0.999   | 0.999  | 1.0    | 1.0   |



# Resultados da Entropia por estratégia nos dados de teste (9k)

[illegible]

# Considerações preliminares

- ◉ Existem muitas coisas baseadas em entropia
- ◉ Frequentemente usadas para análise de séries temporais
- ◉ Os pontos discutidos até aqui possibilitariam implementar um árvore de decisão (CART) melhorada em relação ao existe (outro foco)
- ◉ Nós podemos usar a distribuição de probabilidades de um atributo  $p(x_i)$  usando divergência de Kullback–Leibler.
  - ◉ A divergência de Kullback–Leibler permite medir a distância entre duas distribuições ou a perda de informação entre as duas distribuições.