Exercicio Programa № 1

Instituto de Matematica e Estatistica - Universidade de São Paulo (IME-USP) March 26, 2019

1 Introdução

Esse exercício programa (EP) está dividido em duas partes. A primeira parte consiste em um programa do tipo "console" que solicitará informações para o usuário por meio de um menu e o usuário irá interagir com esse programa via o teclado utilizando um "terminal". A segunda parte consiste da utilização de Jupyter notebook (https://jupyter.org/), onde você deverá gerar um relatório dos dados "coletados" na parte 1 do exercício.

O objetivo desse EP é avaliar e praticar conceitos básicos de programação utilizando Python: condicionais, repetição, funções, classes, criação e importação de módulos, leitura e escrita de arquivos. Também será praticado os conceitos inciais vistos nas primeiras semanas de aula da disciplina: conjunto de treinamento, plotagem de espalhamento, plotagem de reta, Perceptron.

2 Pré-requisitos

Esse exercício necessita da instalação dos seguintes softwares e bibliotecas Python:

- Python 3.x (https://www.python.org/): Linguagem de programação Python.
- Jupyter notebook (https://jupyter.org/): Ambiente para criação e utilização de documentos com *live code*, equações e visualizações.
- NumPy (http://www.numpy.org/): Biblioteca Python para computação cientifica.
- Matplotlib (https://matplotlib.org/): Biblioteca Python para plotagem.
- Pandas (https://pandas.pydata.org/): Biblioteca Python para manipulação de dados.
- Tabulate (https://github.com/gregbanks/python-tabulate): Biblioteca para imprimir tabelas formatadas no terminal.

Embora seja possível instalar esses softwares e bibliotecas manualmente, recomendamos utilizar um pacote de instalação como o Anaconda (https://www.anaconda.com/distribution/#download-section) que além de instalar Python em um ambiente isolado do sistema ainda instala todas as bibliotecas comuns para se trabalhar com dados e Machine Learning em Python.

Se optar por utilizar o Anaconda todos softwares e bibliotecas necessárias para esse exercício serão instaladas exceto a tabulate que por sua vez pode ser instalada no seu ambiente do Anaconda utilizando o comando (https://anaconda.org/conda-forge/tabulate):

```
conda install -c conda-forge tabulate
```

3 Parte 1 - Programa de cadastro de Iris

Vamos supor que existe uma loja que vende dois tipos de flores de Iris: (i) Iris-versicolor e (ii) Iris-setosa. Essa loja encomendou um sistema de *Machine Learning* que diz qual tipo de flor de Iris dadas medidas de largura (*width*) e comprimento (*length*) das sépalas. Para isso, a loja solicitou um aplicativo "console" para o cadastro (coleta) de dados de algumas flores de iris. Esse aplicativo está parcialmente desenvolvido e sua tarefa nessa primeira parte do exercício é terminar de desenvolver esse aplicativo.

3.1 Passo 1 - Imprimir uma mensagem de início

Em Python, o comando `print` é utilizado para imprimir uma mensagem no "console". Por exemplo, diversos tutoriais de programação sempre iniciam com o programa que imprime "hello world" no console. O programa "Hello World" em Python é simplesmente:

```
print("Hello World.")
```

Para executar esse programa você deve abrir um novo arquivo em um editor de texto (bloco de notas, Atom, Sublime, GEdit, etc) e digitar esse programa e salva-lo com a extensão ".py". Por exemplo, se o arquivo foi salvo com o nome "hello.py", então para executar o programa você deve abrir um terminal, ir até à pasta em que o arquivo foi salvo e executar o interpretador Python:

```
python3 hello.py
```

Sua primeira tarefa nesse exercício será imprimir no console a seguinte mensagem:

```
"========= Programa de Cadastro de Flores de Iris ==============="
```

antes da impressão do menu inicial que ocorre na linha:

```
  app = get_app_from_options_menu()
```

do arquivo "irisapp.py".

3.2 Passo 2 - Leitura de dados pelo terminal e funções

Uma *função* é um importante conceito em programação utilizado para a organização do programa e reutilização de código. Nesse exercício, você terá que escrever uma função que lê dados do usuário na aplicação console. Antes de introduzir funções, é importante conhecer o conceito de *escopo*. Escopo são blocos de código delimitados, o código escrito em um escopo é valido apenas dentro desse escopo. Um escopo em Python é denotado por ":" (dois-pontos) e endentação (espaçamento do código até a margem) de um TAB. Uma função em Python, é composta por seu nome, uma lista de parâmetros e um bloco de código dentro de seu escopo. Por exemplo, uma função que soma dois números em Python pode ser escrita como:

```
1 def sum(n1, n2):
2     r = n1 + n2
3     return r
```

então podemos *chamar* essa função posteriormente no código, por exemplo:

```
1 print(sum(4,9))   # chama sum, imprime "13"
2 x1 = 15
3 x2 = 5
4 r = sum(x1, x2) # chama "sum"
5 print("resultado: ", r) # imprime "resultado: 20"
```

Uma função que não resulta em um valor (ou objeto) não terá um comando de retorno (`return`) e funções que não recebem parâmetros possui uma lista vazia um exemplo é a função `print_menu` já codificado no código do exercício.

O comando (função) para ler dados do usuário é o comando `input`. Por exemplo, o programa que soma 5 a um valor dado pelo usuário pode ser escrito:

```
1 valor = int(input("Entre com o valor a ser somando: ")) # leitura de dados
2 print("O resultado da soma eh: ", sum(valor, 5))
```

Um importante detalhe da função `input` é que ela retorna uma "string" (tipo de dados que representa uma sequência de caracteres alfanuméricos) que o usuário digitou, por isso, para podermos somar com um número inteiro é preciso converter o valor de retorno para inteiro utilizando a função `int` conforme exibido no exemplo.

Neste exercício, é pedido que você desenvolva uma função em linguagem Python. Essa função deve: (i) ser nomeada `read_option`, (ii) não receber parâmetro algum e (iii) devolver um número inteiro. Essa função deve imprimir um prompt dizendo `"Entre com a opção: "` solicitando um número inteiro que foi digitado pelo usuário.

Existe um comentário no código (arquivo `irisapp.py`) sugerindo um lugar para criar essa função. Chame essa função e atribua o resultado a variável `option` na função `get_app_rom_option_menu` no local indicada pelo comentário. Para isso substitua o trecho

`option = 0` pela chamada de sua função é atribua o resultado ao `option`.

3.3 Passo 3 - Condicionais, implementando funcionamento do menu

Condicionais são estruturas de programação utilizadas para o que o programa possa tomar decisões de acordo com certas condições lógicas. Por exemplo, suponha que desejamos escrever um programa que recebe a média de aluno e diz se ele foi "aprovado", "para exame" e "reprovado" então podemos escrever um programa da seguinte forma:

```
1 media = calcula_media()
2 if media < 3.5:
3    print("reprovado")
4 elif 3.5 <= media and media < 5:
5    print("exame")
6 else:
7    print("aprovado")
```

Nesse exercício, é pedido que você implemente o funcionamento do menu inicial. Ao iniciar o programa, serão apresentadas três opções:

```
1 ================ Menu ====================
2 (1) carregar arquivo
3 (2) criar arquivo
4 (0) sair
```

E se o passo 2 foi implementado, após o menu ser impresso será solicitado para o usuário informar a opção desejada. Se ele entrar com o número 1 você deve chamar e retornar o resultado da função `load_file`. Se ele entrar com o número 2 você deve chamar e retornar o resultado da função `new_file` e se o usuário entrar com o número 0 você deve chamar e retornar o valor da função `exit`.

3.4 Passo 4 - Introdução as classes, representação de flor de Iris

Nesse exercício, serão praticados o conceito de classe e módulos em linguagem Python. Módulos são utilizados para organizar o código de programas grandes em diferentes arquivos e pastas assim como a escrita de código reusável em outros programas. Algumas bibliotecas que usaremos no curso são organizadas em diversos módulos, por exemplo, NumPy, Pandas, Matplotlib e Scikit-learn. Um modulo Python é simplesmente um arquivo de código Python com declarações e definições. O exercício possui três módulos: "app", "dataset" e "io" (todos dentro da pasta iris). Nesse passo, é pedido que você modifique um desses três módulos. Primeiramente, você deverá implementar a classe "Iris" no modulo "dataset". Classe é o recurso principal de Programação Orientada a Objetos (POO) e é utilizada para juntar dados e operações relacionadas em uma única estrutura. Classe é um conceito importante em progra-

mação porque fornece uma maneira de organização de código onde uma classe depende da outra e um programa grande pode ser divido em diferentes classes que se ligam entre si. No exercício, o modulo "dataset" possui duas classes: (i) `IrisType` e (ii) `Dataset`. Uma classe é definida pelo seu conjunto de atributos e métodos. Os *métodos* são funções que sempre são chamadas no contexto de uma instância da classe. Ou seja, os métodos necessitam que uma variável (ou objeto) do tipo da classe seja criada. Esse contexto é acessado pelo primeiro parâmetro "self" que deve ser declarado em toda definição de método. Por exemplo, a classe `Dataset` possui diversos métodos, dois exemplos são `to_list_header` que retorna uma lista contendo o cabeçalho da tabela e `csv_header` que retorna uma *string* com o cabeçalho para um arquivo *csv*:

```
1  class Dataset:   # Definicao da classe
2    # ... Varios metodos
3
4    # metodo to_list_header
5    def to_list_header(self):
6        return ["sepal width", "sepal length", "type"]
7
8    # metodo csv_header
9    def csv_header(self):
10       return "sepal width;sepal length;type"
```

Para chamar esses métodos é necessária uma instância da classe `Dataset`, conforme mostrado abaixo:

```
1  d = Dataset()    # cria-se uma instancia da Dataset
2  print(d.to_list_header()) # chama metodo to_list_header no contexto de "d"
```

Os atributos de uma classe são os dados que podem ser armazenados por suas instâncias. Uma instância de uma classe é caracterizada pelos valores de seus atributos e o conjunto desses valores geralmente é referenciado como o *estado do objeto*. Por exemplo, no código do exercício, são definidas duas instâncias da classe `IrisType` (classe que representa tipos de Iris) conforme exibido abaixo:

```
1  IrisVersicolor = IrisType(id=0, name="Iris-Versicolor")
2  IrisSetosa = IrisType(id=1, name="Iris-Setosa")
```

A instância `IrisVersicolor` possui os valores "Iris-Versicolor" como `name` e $"0"$ como identificador (`id`) e a instância `IrisSetosa` possui os valores "Iris-Setosa" como `name` e $"1"$ como identificador (`id`). Então podemos imprimir os valores de cada instância com o seguinte código:

```
1  print("nome: ", IrisVersicolor.name, " identificador: ", IrisVersicolor.id↩
```

```
    )
print("nome: ", IrisSetosa.name, " identificador: ", IrisSetosa.id)
```

Essas linhas de código iram imprimir no *terminal*:

```
nome: Iris-Versicolor, identificador: 0
nome: Iris-Setosa, identificador: 1
```

Geralmente as classes têm um método especial chamado de construtor. O construtor é método que é chamado quando uma instância da classe é criada e geralmente é utilizado para inicializar (estado inicial) os atributos da instância. O construtor é criado quando se define um método chamado `__init__`. Por exemplo, no código do exercício os construtores das classes `IrisType` e `Dataset` são definidos como:

```
class IrisType:
  def __init__(self, id, name):
      self.id = id
      self.name = name
# ...
class Dataset:
  def __init__(self):
    self.data = []
```

O construtor da classe `IrisType` simplesmente cópia os parâmetros para os atributos da classe e o construtor da classe `Dataset` apenas inicializa o atributo `data` com uma lista vazia.

Nesse passo, você deve implementar uma classe no modulo "`dataset`" chamada `Iris` essa classe representa uma flor de iris por meio de três atributos: (i) `sepal_width` (largura da sépala) do tipo `float`, (ii) `sepal_length` (comprimento da sépala) do tipo `float` e (iii) `type` (tipo) do tipo `IrisType`. Crie um construtor que copie os parâmetros para cada um dos atributos e dois métodos: (i) `attr_to_list` que retorna uma lista com os valores dos atributos em uma lista na seguinte ordem: `sepal_width`, `sepal_length` e o nome do `type` (atributo `name`) e (ii) `to_csv_line` que retorna uma string que representa um registro de flor de iris em um arquivo csv (valores separados por ponto-vírgula ";") na seguinte ordem: `sepal_width`, `sepal_length` e o identificadr do `type` (atributo `id`).

3.5 Passo 5 - Importação de classes, funções e leitura e escrita de arquivos

Nesse exercício será implementado as funcionalidades de ler e escrever dados armazenados em uma instância da classe `Dataset` em arquivos "csv". Também será abordado o funcionamento de módulos em Python. No exercício, além do modulo "`dataset`" onde foi implementado a classe `Iris` no exercício anterior também existe o modulo `io` criado para fornecer funcionalidades de escrever arquivo "csv" de dataset de Iris em arquivo e ler esse arquivo para memória em um objeto do tipo `Dataset`. Um modulo Python é um arquivo de código em Python com

declarações e definições como foi feito no exercício anterior. Quando criamos um arquivo de código Python com um programa e queremos utilizar as definições e declarações de um modulo Python, precisamos primeiro *importar* esse modulo. Um modulo Python é sempre importado considerando o caminho relativo do arquivo para o local onde será chamado o interpretador Python. Por exemplo, no exercício, o programa principal "`irisapp.py`" está na pasta "raiz" (".") localizada em uma pasta qualquer e os arquivos estão organizados da seguinte maneira:

- "."
  - `irisapp.py`
  - "iris"
    - `app.py`
    - `dataset.py`
    - `io.py`

Nesse exemplo (e no exercício), os nomes entre aspas representam diretórios, os espaçamentos indicam a hierarquia de pastas e arquivos. Nosso programa está em `irisapp.py` e o interpretador Python sempre será chamado nessa pasta. Por isso se quisermos importar o modulo dataset no modulo `io` podemos importa-lo e utiliza-lo da seguinte maneira:

```
1 import iris.dataset  # importa modulo dataset
2 d = iris.dataset.Dataset()
3 it = iris.dataset.IrisVersicolor
```

Como escrever o nome do modulo ("`iris.dataset`" no nosso exemplo) toda vez que for utilizar uma de suas definições pode ser massante, Python permite criar alias (pseudônimo ou apelido) para o modulo. Por exemplo, um programa com a mesma funcionalidade poderia ser escrito como:

```
1 import iris.dataset as dt  # importa modulo dataset com alias "dt"
2 d = dt.Dataset() # Agora eh possivel trocar o nome do modulo pelo alias
3 it = dt.IrisVersicolor
```

Python permite também importar apenas as definições que irão ser utilizados na sua aplicação. Isso evita o carregamento de código desnecessário melhorando o desempenho em memória e processamento. Utilizando esse recurso, podemos reescrever essa aplicação da seguinte forma:

```
1 # a linha debaixo importa Apenas as definicoes de Dataset e IrisVersicolor
2 from iris.dataset import Dataset, IrisVersicolor
3 d = Dataset()
4 it = IrisVersicolor
```

Utilizando essa forma de importar as definições de modulo precisamos tomar cuidado para que o nome que estamos importando não se choque com nenhum outro nome (por exemplo, imagine que existam classes `Dataset` importadas de diferentes módulos), utilize sempre um alias quando isso ocorrer.

Nesse passo, a primeira coisa que você deve fazer é importar as definições de `Dataset`, `IrisVersicolor`, `IrisSetosa` e `Iris` do modulo `iris.dataset`. Também é pedido para implementar a funcionalidade de lêr um arquivo ".csv" gerado pelo programa do exercício do arquivo para memoria (em um objeto do tipo "`Dataset`"). Você pode consultar as funcionalidades de leitura de arquivo em Python pelo link: https://docs.python.org/3/tutorial/inputoutput.html. Resumidamente a leitura de arquivo é feito pela classe de arquivo do Python obtida pela chamada da função `open`. A função `open` recebe como parâmetros: (i) uma `string` com o caminho para o arquivo no disco e (ii) o modo do arquivo (para o caso do exercício, "`w`" para escrita e "`r`" para leitura). A função `open` retorna um objeto do tipo arquivo que depois de seu uso deve ser fechado com o método `close` ou chamado com "`with`" que fecha o arquivo no final do seu escopo, como, por exemplo o código de escrita do arquivo csv disponível no exercício:

```
1 def write_iris_dataset(path, dataset):
2     with open(path, "w") as file:
3         file.write(dataset.csv_header()+"\n")
4         for iris in dataset:
5             file.write(str(iris.sepal_width)+";"+str(iris.sepal_length)+";↩
                  "+str(iris.type.id)+"\n")
```

Nesse trecho de código, primeiro é criado um arquivo utilizando "`with`" no caminho "`path`" e modo "w" com o nome "`file`". Depois escrevemos no arquivo o cabeçalho do dataset chamando o método "`write`" do arquivo e `csv_header` do `dataset`. Então para cada elemento que está no dataset, é escrito uma linha do registro do `csv` (separando cada coluna com um ponto-e-virgula ";"). Como foi utilizado o `with` ao final do método o arquivo é fechado automaticamente.

Nesse exercício, é pedido que você implemente uma função chamada `read_iris_dataset` que recebe como parâmetro um caminho de arquivo csv gerado pela função "`write_iris_dataset`". Você deve preencher uma instância da classe `Dataset` com objetos do tipo `Iris` com os dados do arquivo "csv". Retorne o objeto `Dataset` preenchido.

3.6 Passo 6 - Implemente a função de leitura de arquivo

Implemente a função `load_file` utilizando a função `read_iris_dataset`. Primeiro você terá que solicitar um caminho de arquivo csv para usuário. Depois você deve ler esse caminho e preencher um objeto `Dataset` com os dados do arquivo, retorne uma instância `IrisApplication` com o `dataset` e `filepath` preenchidos.

3.7 Passo 7 - Implementação da classe IrisApplication

Utilize o código apresentado até o momento para terminar de implementar a classe `IrisApplication`. A classe `IrisApplication` possui o construtor, mais dois métodos implementados (`sort` e `exit`) e um método que você devera implementar nesse passo do exercício (`run_action`). Você irá implementar a principal região da aplicação que deve ter o seguinte menu:

```
1 ==================== MENU ==========================
2 (1) criar registro
3 (2) visualizar banco de dados
4 (3) remover registro
5 (4) ordenar
6 (5) salvar banco de dados em arquivo
7 (0) sair
```

Então seu aplicativo deve solicitar a entrada de um número inteiro de (0 a 5) ao usuário. Abaixo, mostra as ações que seu aplicativo deve tomar a cada número que usuário entrar:

- (1): Seu programa deve solicitar valores para "sepal length" (comprimento da sépala), "sepal width" (largura da sépala), e "identificador de iris type" (identificador do tipo de flor de iris). A partir dos dados informados pelo usuário, você deve criar uma instância da classe `Iris` e adiciona-la ao seu "`dataset`" e então retornar ao menu.
- (2): Seu programa deve exibir uma tabela com todos os registros de Iris em memória (que estão armazenados no atributo `dataset` e então retornar para o menu. SUGESTÃO: A biblioteca Tabulate pode ser útil para gerar uma tabela bem formatada.
- (3): Seu programa deve solicitar um índice (linha na tabela de visualização começando do zero) e remover essa linha do seu `dataset`. Então retornar ao menu.
- (4): Solicitar um campo para ordenação e então ordenar os registros `dataset` então volte ao menu. SUGESTÃO: O método `sort` já disponível pode te ajudar com esse exercício.
- (5): Salvar os registros do `dataset` em um arquivo csv especificado pelo atributo `filepath` já disponível na classe `IrisApplication`. Então retorne ao menu.
- (0): Encerre o programa.

Para implementar esse exercício, note que no seu arquivo principal (que será utilizado como entrada para o interpretador Python) "`irisapp.py`" existe um laço que se encerra quando o atributo da instância de `IrisApplication` possui o atributo `should_exit` com valor `True`. Além disso, a cada iteração desse laço é executado o método `run_action` que você deve implementar na classe `IrisApplication`. Para isso, você deve apagar o comando "`pass`" e implementar o seu código nesse método.

3.8 Passo 8 - geração do dataset de treinamento e testes

Nesse passo do exercício, você utilizará o aplicativo desenvolvido para gerar o seu conjunto de dados de treinamento e de testes. Para o conjunto de treinamento, utilize como dataset inicial o arquivo "treino.csv" e inclua no arquivo os seguintes registros:

sepal width	sepal length	type
5.0	3.4	Iris-setosa
4.4	2.9	Iris-setosa
4.9	3.1	Iris-setosa
5.4	3.7	Iris-setosa
4.8	3.4	Iris-setosa
7.0	3.2	Iris-versicolor
6.4	3.2	Iris-versicolor
6.9	3.1	Iris-versicolor
5.5	2.3	Iris-versicolor
6.5	2.8	Iris-versicolor

Para gerar o dataset de testes, você deve gerar um arquivo com o nome "teste.csv" (criando o arquivo com o aplicativo desenvolvido nesse exercício). Gere esse arquivo cadastrando os dados da tabela abaixo:

sepal width	sepal length	type
5.0	3.0	Iris-setosa
5.0	3.4	Iris-setosa
4.5	2.3	Iris-setosa
5.2	3.4	Iris-setosa
4.7	3.2	Iris-setosa
4.8	3.1	Iris-setosa
6.0	2.2	Iris-versicolor
6.1	2.9	Iris-versicolor
5.6	2.9	Iris-versicolor
6.7	3.1	Iris-versicolor
5.6	3.0	Iris-versicolor
5.8	2.7	Iris-versicolor

4 Parte 2 - Implementação do Perceptron e plotagem de dados

Jupyter notebook é uma ferramenta que nos permite misturar código Python e texto de uma forma estruturada. O Jupyter notebook é um ambiente dividido em células. Cada célula pode ser de texto (`markdown`) ou de código (geralmente Python). Para trocar o tipo de célula basta clicar no combo box na barra de ferramentas (deixe na opção "`code`" para código e "`markdown`"

para texto) ou no modo de navegação utilizar as teclas "M" para markdown ou "Y" para código.

Nessa parte, você primeiramente precisa copiar os arquivos de dados gerados no exercício anterior (`treino.csv` e `teste.csv`) no diretório `report`. Depois no terminal ir até à pasta `report` e executar o Jupyter notebook no arquivo ("`perceptron.ipynb`):

```
1 $ cd report
2 $ jupyter notebook
```

Nesse ponto basta escolher o arquivo `perceptron.ipynb`. Os enunciados e as células onde você deve desenvolver o código está anotado no Jupyter notebook.