Professor Rodrigo Bossini
https://sites.google.com/site/professorrodrigobossini/

Data Science com Python 3
K-Vizinhos mais próximos

Imagine que desejamos prever como uma pessoa irá votar nas próximas eleições presidenciais. Não sabemos nada sobre essa pessoa mas sabemos onde ela mora. Uma possibilidade para tentar prever seu voto é verificar os votos de seus vizinhos. É relativamente natural que pessoas que moram perto tenham gostos parecidos. Agora imagine que também sabemos outras coisas como idade, número de filhos e renda. É ainda mais natural considerar apenas as intenções de voto de pessoas que tenham similaridade nessas variáveis para tentar fazer a predição. O algoritmo KNN (K Nearest Neighbors) requer duas coisas:

- Uma noção de distância
- Uma premissa de que pontos próximos entre si são similares

A ideia é encontrar os K pontos mais próximos e utilizar seus valores e intenções de voto para fazer a predição para a pessoa cuja intenção de voto é desconhecida.

Passo 1 (Identificando os K pontos mais próximos e encontrando os valores mais frequentes) Suponha que escolhemos um valor para K, como 3 ou 5. Para realizar a predição de uma variável para um novo ponto, verificamos quem são os K pontos mais próximos e verificamos, dentre eles, qual valor da variável cujo valor desejamos prever é o mais comum.

1.1 Veja os exemplos da Tabela 1.1.

Tabela 1.1 – Pesquisa para eleições presidenciais

	Variáveis				
	idade	renda mensal	sexo	intenção de voto (classe ou rótulo)	
K = 3	27	2500	M	Bolsonaro	Valor previsto: Bolsonaro
	28	3000	F	Haddad	
	26	5000	M	Bolsonaro	
K = 5	27	2500	M	Bolsonaro	Valor previsto: Haddad
	28	3000	F	Haddad	
	26	5000	M	Bolsonaro	
	37	6000	M	Haddad	
	45	10500	F	Haddad	

Note que os exemplos da Tabela 1.1 usam o valor mais comum para determinar a predição para um determinado valor de K. Porém, há alternativas, como veremos adiante.

Passo 2 (Uma base de dados para exemplo) A função gera_base da Listagem 2.1 gera uma pequena base de dados para os testes. Ela é composta por objetos do tipo Pessoa.

Listagem 2.1

```
class Pessoa:
    def __init__ (self, idade, sexo, salario, intencao_de_voto):
        self.idade = idade
        self.sexo = sexo
        self.salario = salario
        self.intencao_de_voto = intencao_de_voto
    def __str__ (self):
        return f'idade: {self.idade}, sexo: {self.sexo}, salario: {self.salario},
intencao_de_voto: {self.intencao_de_voto}'

def gera_base (n):
    l = []
    for i in range (n):
        #aleatorio no intervalo [a, b]
        idade = random.randint(18, 35)
        sexo = random.choice(['M', 'F'])
        # aleatorio no intervalo [0.0, 1.0)
        salario = 1200 + random.random() * 1300
        intencao_de_voto = random.choice(['Haddad', "Bolsonaro"])
        p = Pessoa (idade, sexo, salario, intencao_de_voto);
        l.append(p)
    return l
```

Passo 3 (Escrevendo uma função para identificar os rótulos mais frequentes) A função da Listagem 3.1 recebe a coleção de classes ou rótulos que pertencem às K instâncias mais próximas, as quais desejamos utilizar na predição da classe de uma nova observação. Note que sua implementação depende da sobrescrita dos métodos __eq__e __hash__ da classe Pessoa.

Listagem 3.1

```
def rotulo_de_maior_frequencia (pessoas):
    #frequência dos rótulos
    frequencias = Counter ([pessoas])
    #devolve uma lista de tuplas
    #primeiro elemento: rotulo
    #segundo elemento: frequencia
    #se houver desempate, desempata arbitrariamente
    mais_frequentes = frequencias.most_common(1)
    #retira a tupla da lista e pega somente o rotulo
    return mais_frequentes[0][0]

#na classe Pessoa
def __eq__ (self, other):
```

```
    return self.intencao_de_voto == other.intencao_de_voto
def __hash__(self):
    return 1 #ineficiente
```

Passo 4 (Lidando com observações com frequência igual) Muitas vezes pode acontecer de, entre os K vizinhos mais próximos, existirem observações com rótulo diferente e frequência igual. Neste caso, qual a melhor classificação a ser utilizada? Há diferentes estratégias, como:

- Escolher um aleatoriamente
- Calcular o somatório de distâncias para cada rótulo e utilizar o que resultar em menor valor final
- Reduzir o valor de k e encontrar um único vencedor

A função da Listagem 4.1 implementa a terceira estratégia.

Listagem 4.1

```
def rotulo_de_maior_frequencia_sem_empate (pessoas):
    frequencias = Counter (pessoas)
    rotulo, frequencia = frequencias.most_common(1)[0]
    qtde_de_mais_frequentes = len([count for count in frequencias.values() if count ==
frequencia])
    if qtde_de_mais_frequentes == 1:
        return rotulo
    return rotulo_de_maior_frequencia_sem_empate(pessoa[:-1])
```

Passo 5 (Testando o que temos até então) A Listagem 5.1 define um primeiro teste.

Listagem 5.1

```
def main ():
    base = gera_base(10)
    print (rotulo_de_maior_frequencia_sem_empate(base))
main ()
```

Passo 6 (A função de distância) A função de distância que utilizaremos calcula a distância Euclidiana entre dois pontos. Veja a Listagem 6.1.

Listagem 6.1

```
def distance (p1, p2):
    i = math.pow((p1.idade - p2.idade), 2)
    s = math.pow((1 if p1.sexo == 'M' else 0) - (1 if p2.sexo == 'M' else 0), 2)
    sal = math.pow((p1.salario - p2.salario), 2)
    return math.sqrt(i + s + sal)
```

Passo 7 (O algoritmo KNN) Agora podemos implementar o algoritmo KNN. Ele recebe o valor de K desejado, o conjunto de observações cuja classe é conhecida e um ponto a ser classificado. Veja a Listagem 7.1.

Listagem 7.1

```
def knn (k, observacoes_rotuladas, nova_observacao):
    ordenados_por_distancia = sorted (observacoes_rotuladas, key= lambda obs: distance (obs,
nova_observacao))
    k_mais_proximos = ordenados_por_distancia[:k]
    resultado = rotulo_de_maior_frequencia_sem_empate(k_mais_proximos)
    return resultado.intencao_de_voto
```

A Listagem 7.2 mostra um teste para o algoritmo KNN.

Listagem 7.2

```
def main ():
    base = gera_base(10)
    for p in base:
        print (p)
    print(knn(3, base, Pessoa (19, 'M', 1700, None)))
main ()
```

Bibliografia

GRUS, J. **Data Science from Scratch: First Principles with Python.** 1st ed. O'Reilly Media, 2015.