

RF-based Direction Finding of UAVs Using Deep Neural Network

Anabel Reyes Carballeira (anabel.carballeira@mtel.inatel.br)

Eligário Milton da Costa Semedo (eligario@mtel.inatel.br)

National Institute of Telecommunications – Inatel

Master in Telecommunications

1st Semester, 2020

Introdução



- Violação de privacidade
- Ações criminais: roubo ou invasão domiciliar
- Tráfico de drogas dentro e fora das prisões
- Colisão e queda de drones nas pessoas.

- Em janeiro de 2015, um drone caiu na Casa Branca



- Em março de 2016, um jato da Lufthansa chegou a 200 pés de colisão com um drone perto do Aeroporto Internacional de Los Angeles



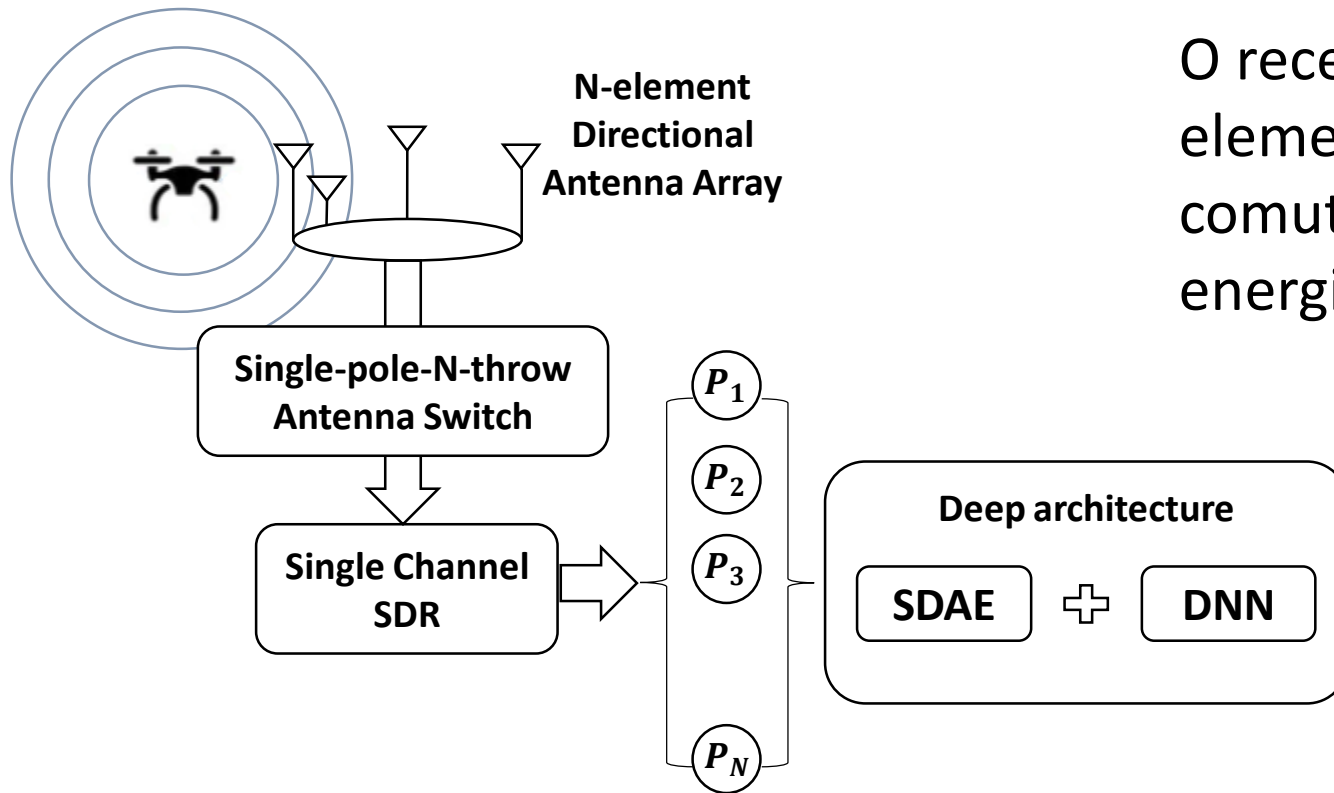
Introdução

- **As técnicas de detecção de direção (DF) baseadas em Radiofrequência foram bem estudadas e as técnicas clássicas de alta resolução, como MUSIC e ESPRIT.**
- **Uso de comunicação WiFi como o OFDM (que são as técnicas mais usadas para comunicação por drones) que apresentam alguns desafios bem como a complexidade do hardware e o consumo de energia.**

Objectivo

Propomos um método **prático e de baixo nível de complexidade** para **encontrar a direção** do drone (DF)

Sistema proposto para localização de direção de UAVs baseados em RF



O receptor ativa seqüencialmente um elemento de antena de cada vez usando o comutador RF-SPNT e mede o valor de energia recebido correspondente.

SDAE: sparse denoising autoencoder
DNN: deep neural network

Goal

Obter um relacionamento subjacente (ou um padrão) entre a potência média do sinal recebido no conjunto da n-ésima antena $\{\mathbf{P}_n\}_{n=1}^N$ e o ângulo do azimuth θ .

Algoritmo SDAE

1. Durante a fase de treinamento do SDAE, os valores de potência recebidos pré-processados **são atribuídos às unidades de entrada**.
2. Em seguida, os valores das unidades **da camada oculta e da camada de saída** são calculados como:

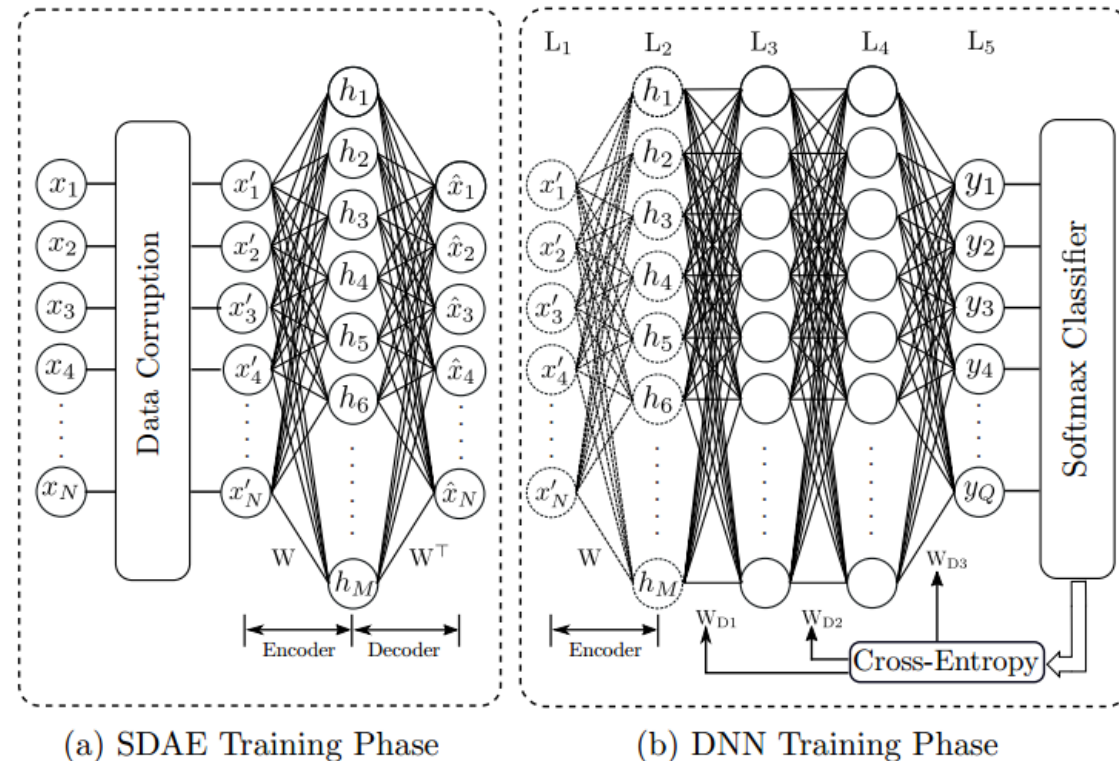
$$h = f(Wf_c(x) + b_e) \text{ e } x_{\text{output}} = f(W^T h + b_d)$$
3. Os parâmetros de SDAE (W , b_e e b_d) **são otimizados** de forma que o erro de reconstrução seja minimizado, enquanto está sujeito a uma restrição de esparsidade.

$$L(W, b_e, b_d) = \sum_{i=1}^T (\hat{x}_i - x_i)^2 + \beta \sum_{m=1}^M \text{KL}(\rho || \rho_m),$$

$$\rho_m = \frac{1}{T} \sum_{i=1}^T h_m(x_i)$$



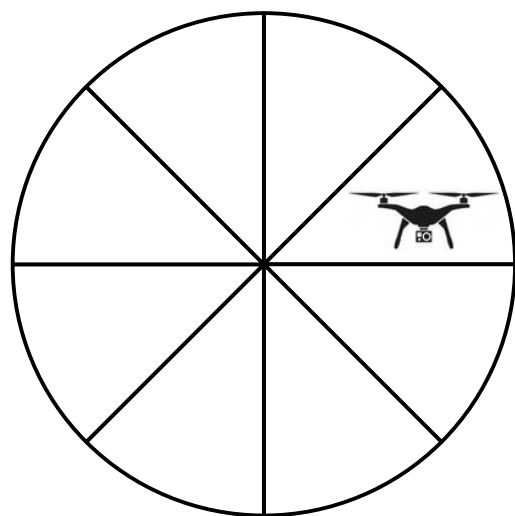
TensorFlow



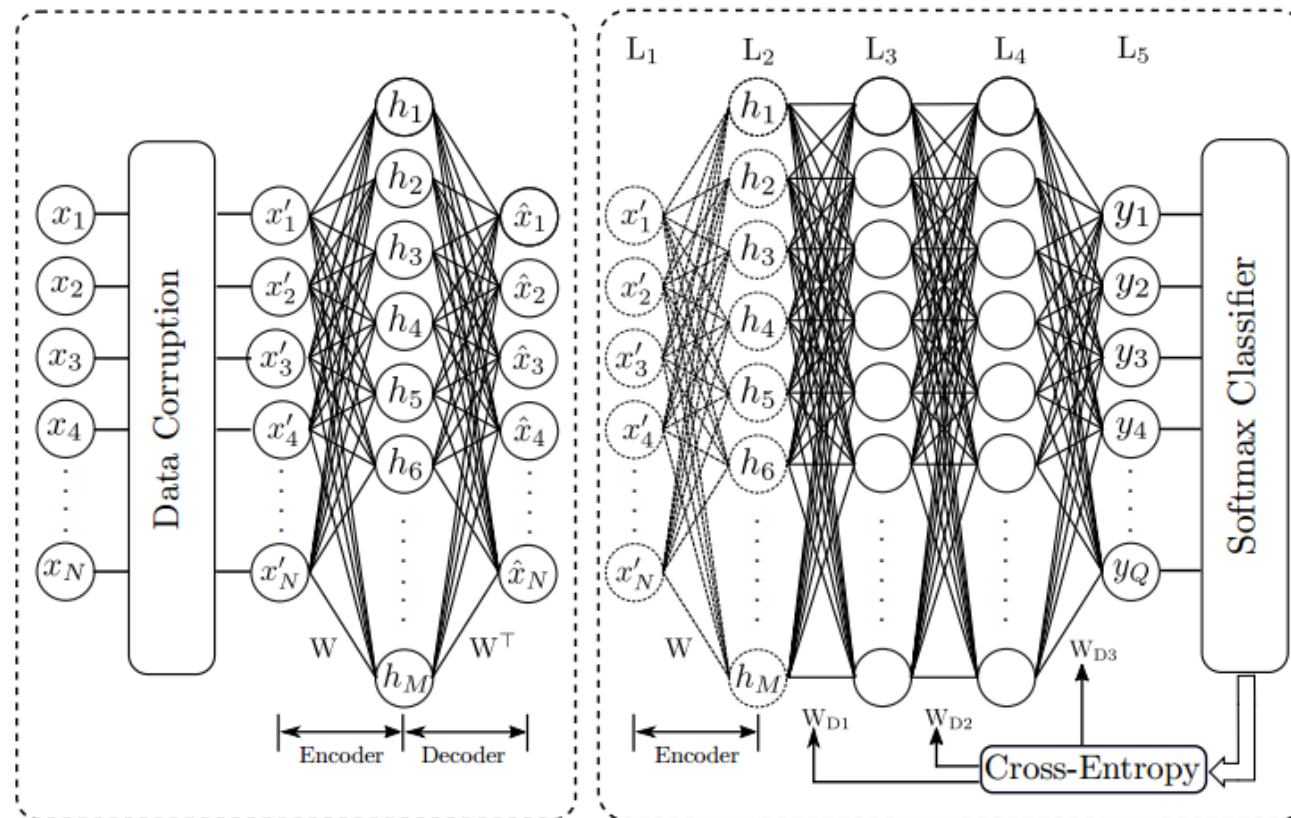
Algoritmo DNN

1. L2 usa a mesma função de ativação **f**.
2. As camadas ocultas L3 e L4 usam a **Unidade de revestimento retificado (ReLU)** como sua função de ativação.
3. Os dados **precisam ser rotulados nas classes Q** devido ao uso do classificador softmax, onde o rótulo é a direção do sinal do drone vindo.

Validação experimental



Top view



(a) SDAE Training Phase

(b) DNN Training Phase

L5- oito neurônios (temos **oito classes** para a classificação de direção).

L1- quatro neurônios (o conjunto de **antenas tem quatro elementos**, ou, $N = 4$).

L2, L3, L4- 200, 12, 12 e neurônios, respectivamente. Esses valores foram decididos **empiricamente** durante o processo de treinamento.

Considerações sobre código e alterações feitas

```
if __name__ == "__main__":
    data = HandleData(total_data=880, data_per_angle=110, num_angles=8)
    antenna_data, label_data = data.get_synthatic_data(test_data=False)

    data_test = HandleData(total_data=80, data_per_angle=10, num_angles=8)
    antenna_data_test, label_data_test = data_test.get_synthatic_data(test_data=True)
```

```
def get_synthatic_data(self, test_data):
```

```
    if test_data is False:
        x_0 = genfromtxt('./Dround_Data_New/Nomalized/deg_0_normalize.csv', delimiter=',', dtype=np.float32)
        x_45 = genfromtxt('./Dround_Data_New/Nomalized/deg_45_normalize.csv', delimiter=',', dtype=np.float32)
        x_90 = genfromtxt('./Dround_Data_New/Nomalized/deg_90_normalize.csv', delimiter=',', dtype=np.float32)
        x_135 = genfromtxt('./Dround_Data_New/Nomalized/deg_135_normalize.csv', delimiter=',', dtype=np.float32)
        x_180 = genfromtxt('./Dround_Data_New/Nomalized/deg_180_normalize.csv', delimiter=',', dtype=np.float32)
        x_225 = genfromtxt('./Dround_Data_New/Nomalized/deg_225_normalize.csv', delimiter=',', dtype=np.float32)
        x_270 = genfromtxt('./Dround_Data_New/Nomalized/deg_270_normalize.csv', delimiter=',', dtype=np.float32)
        x_315 = genfromtxt('./Dround_Data_New/Nomalized/deg_315_normalize.csv', delimiter=',', dtype=np.float32)
    elif test_data is True:
        x_0 = genfromtxt('./Dround_Data_New/Nomalized_test/deg_0_normalize.csv', delimiter=',', dtype=np.float32)
        x_45 = genfromtxt('./Dround_Data_New/Nomalized_test/deg_45_normalize.csv', delimiter=',', dtype=np.float32)
        x_90 = genfromtxt('./Dround_Data_New/Nomalized_test/deg_90_normalize.csv', delimiter=',', dtype=np.float32)
        x_135 = genfromtxt('./Dround_Data_New/Nomalized_test/deg_135_normalize.csv', delimiter=',', dtype=np.float32)
        x_180 = genfromtxt('./Dround_Data_New/Nomalized_test/deg_180_normalize.csv', delimiter=',', dtype=np.float32)
        x_225 = genfromtxt('./Dround_Data_New/Nomalized_test/deg_225_normalize.csv', delimiter=',', dtype=np.float32)
        x_270 = genfromtxt('./Dround_Data_New/Nomalized_test/deg_270_normalize.csv', delimiter=',', dtype=np.float32)
        x_315 = genfromtxt('./Dround_Data_New/Nomalized_test/deg_315_normalize.csv', delimiter=',', dtype=np.float32)
```

Os dados de treinamento e teste já estavam predefinidos e sequencialmente distribuídos.

→ TREINAMENTO

→ TESTE

```
if __name__ == '__main__':
    test_percentage = 0.2

    #instance of the Handle Data class
    data = HandleData(oneHotFlag=False)
    #get the data
    antenna_data, label_data = data.get_synthatic_data()
    antenna_data, antenna_data_test, label_data, label_test = train_test_split (antenna_data, label_data, \
        test_size=test_percentage, random_state=42)
```

```
def get_synthatic_data(self):
```

```
    x_0 = genfromtxt('./Dround_Data_New/Nomalized/deg_0_normalize.csv', delimiter=',', dtype=np.float32)
    x_45 = genfromtxt('./Dround_Data_New/Nomalized/deg_45_normalize.csv', delimiter=',', dtype=np.float32)
    x_90 = genfromtxt('./Dround_Data_New/Nomalized/deg_90_normalize.csv', delimiter=',', dtype=np.float32)
    x_135 = genfromtxt('./Dround_Data_New/Nomalized/deg_135_normalize.csv', delimiter=',', dtype=np.float32)
    x_180 = genfromtxt('./Dround_Data_New/Nomalized/deg_180_normalize.csv', delimiter=',', dtype=np.float32)
    x_225 = genfromtxt('./Dround_Data_New/Nomalized/deg_225_normalize.csv', delimiter=',', dtype=np.float32)
    x_270 = genfromtxt('./Dround_Data_New/Nomalized/deg_270_normalize.csv', delimiter=',', dtype=np.float32)
    x_315 = genfromtxt('./Dround_Data_New/Nomalized/deg_315_normalize.csv', delimiter=',', dtype=np.float32)
```

Alterações

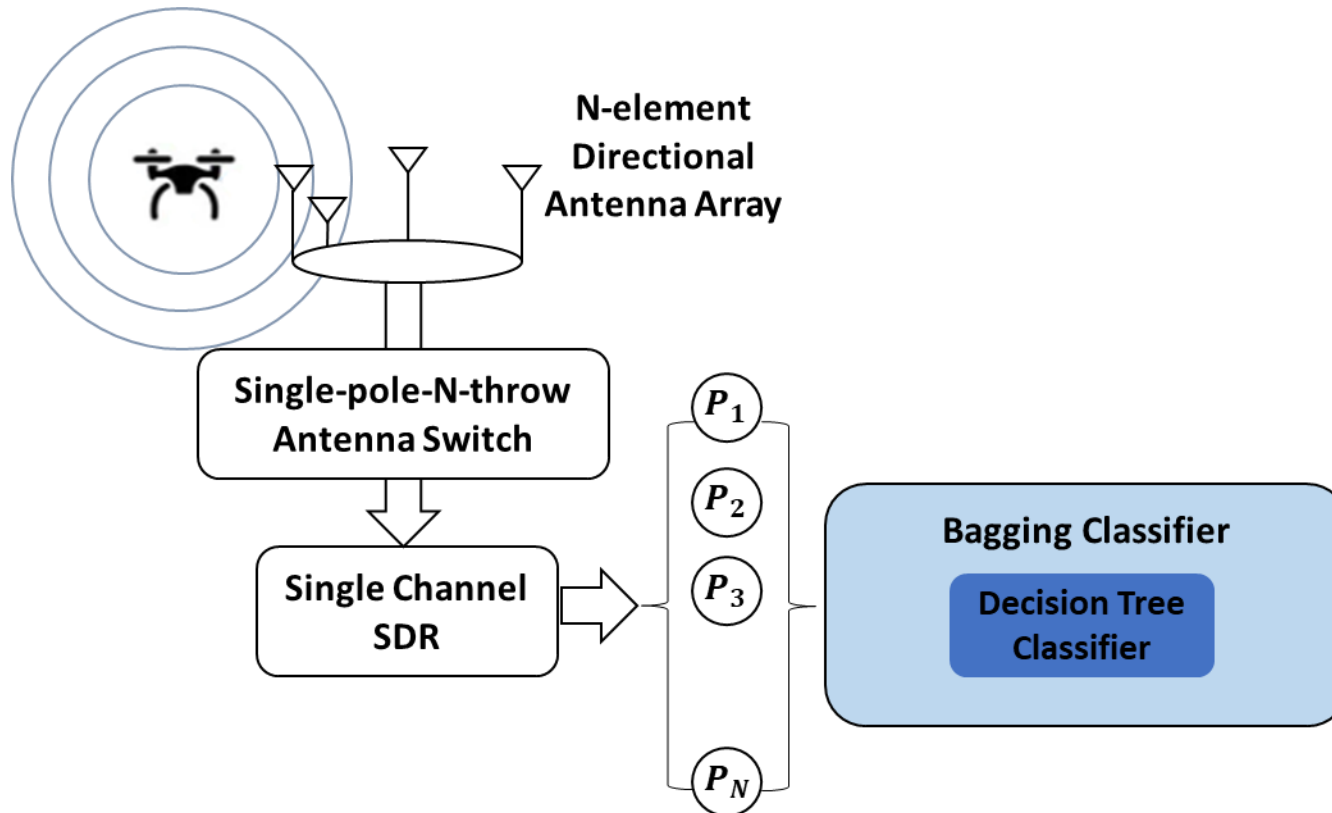
→ TREINAMENTO + TESTE

Outros modelos investigados

- **Árvore de decisão.**
- **Bagging Classifier.**

Outros modelos investigados

- **Árvore de decisão.**
- Bagging Classifier.



Bagging Classifier

```
3 import tensorflow as tf
4 import math
5 import numpy as np
6 from get_csv_data import HandleData
7 import matplotlib.pyplot as plt
8 from sklearn.metrics import confusion_matrix, accuracy_score
9 import seaborn as sns
10 from sklearn.ensemble import BaggingClassifier
11 from sklearn.tree import DecisionTreeClassifier
12 from sklearn.model_selection import train_test_split
13 from time import time

90 if __name__ == '__main__':
91     test_percentage = 0.2

92
93     seed = 1234

94
95     #instance of the Handle Data class
96     data = HandleData(oneHotFlag=False)
97     #get the data
98     antenna_data, label_data = data.get_synthetic_data()
99     antenna_data, antenna_data_test, label_data, label_test =
100         train_test_split(antenna_data, label_data,
101                         test_size=test_percentage, random_state=42)
102
103     # get denoising autoencoder outputs for the train and test data
104     DAE_out = getDAE([antenna_data, antenna_data_test], seed)
105     antenna_data = DAE_out[0]
106     antenna_data_test = DAE_out[1]
107
108     #Instantiate a Bagging Classifier
109     clf = BaggingClassifier(DecisionTreeClassifier(random_state=42), n_estimators=300, max_samples=250, \
110                           bootstrap=False, n_jobs=-1, random_state=42)
111     #Train model
112     tic = time()
113     clf.fit(antenna_data, label_data)
114     #Predict
115     y_pred = clf.predict(antenna_data_test)
```

```
5 class HandleData(object):
6
7     def __init__(self, oneHotFlag=False):
8         self.total_data = genfromtxt('./Dround_Data_New/Nomalized/deg_0_normalize.csv', delimiter=',', \
9                                     dtype=np.float32).shape[0]*len(os.listdir('./Dround_Data_New/Nomalized'))
10        self.data_per_angle = genfromtxt('./Dround_Data_New/Nomalized/deg_0_normalize.csv', delimiter=',', \
11                                        dtype=np.float32).shape[0]
12        self.num_angles = len(os.listdir('./Dround_Data_New/Nomalized'))
13        self.current_point = 0
14        self.data_set = np.zeros((self.total_data, 4), dtype=np.float32)
15        if oneHotFlag == True:
16            self.label_set = np.zeros((self.total_data, self.num_angles), dtype=np.float32)
17        else:
18            self.label_set = [0 for i in range(self.total_data)]
19            self.oneHotFlag = oneHotFlag

39        for i in range(0, self.num_angles):
40            for j in range(0, self.data_per_angle):
41                "add one hot"
42                "add data"
43                if self.oneHotFlag == True:
44                    self.label_set[i * self.data_per_angle + j] = self.onehot_encode(i)
45                else:
46                    self.label_set[i * self.data_per_angle + j] = i
47                    self.data_set[i * self.data_per_angle + j] = data_matrix[i][j]
48
49        return self.data_set, self.label_set
```

0°	45°	90°	135°	180°	225°	270°	315°
0	1	2	3	4	5	6	7

Treina um conjunto de 300 classificadores da Árvore de Decisão.

Cada um treinado em 250 instâncias de treinamento amostradas aleatoriamente do conjunto de treinamento.

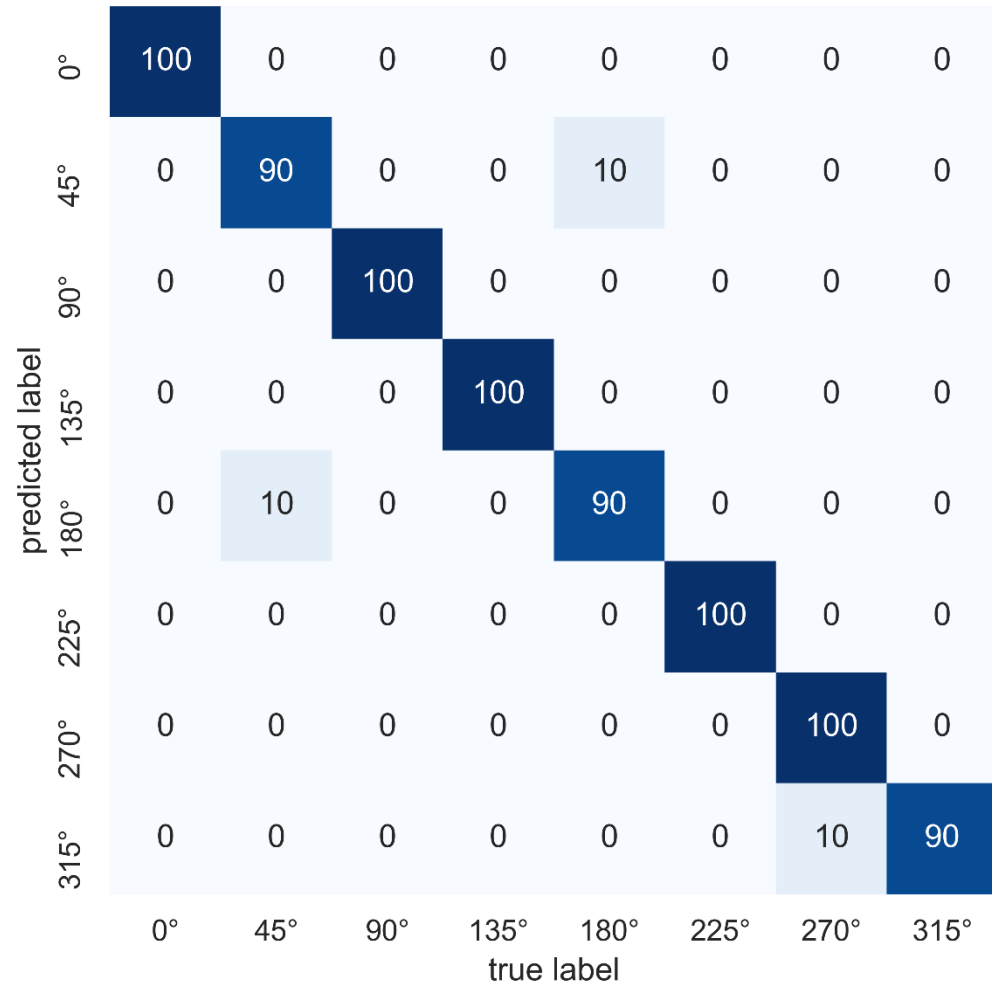
A amostragem sem substituição é realizada.

-1 diz ao Scikit-Learn para usar todos os núcleos do CPU disponíveis.

Comparação dos métodos (8 % dos dados para teste)

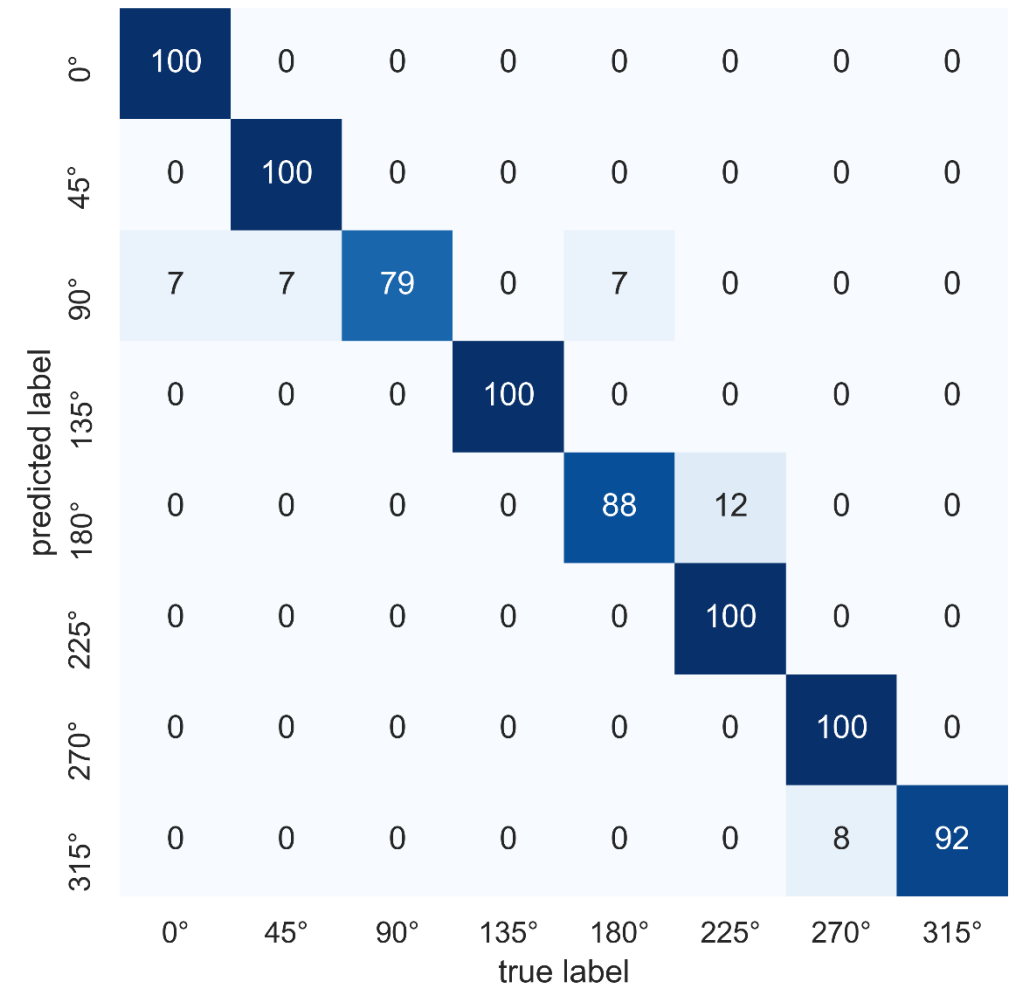
DNN sim modificações

```
Epoch: 2000 cost= 0.119282786  
Optimization Finished!  
Elapsed time: 109.61747765541077 seconds  
Accuracy: 0.96022725  
Accuracy: 0.9625
```



Bagging Classifier

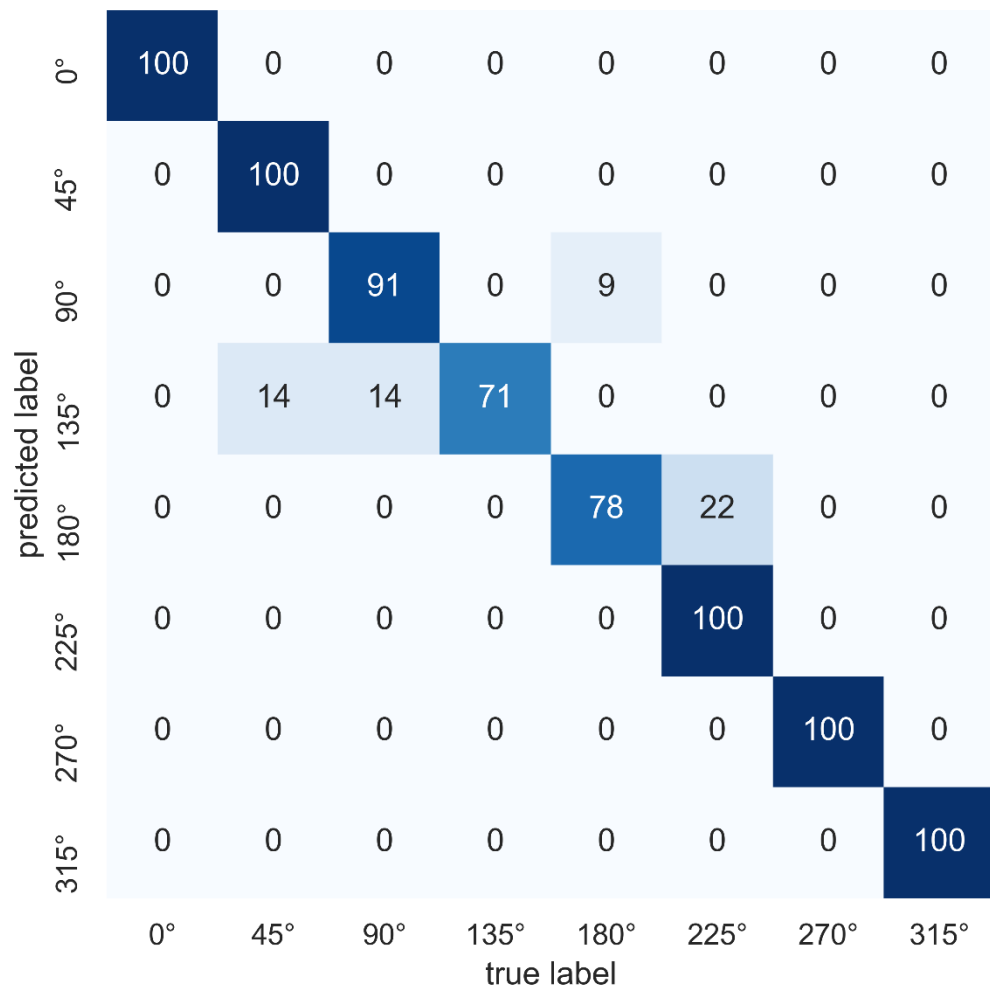
```
Accuracy of model is: 93.5064935064935 %  
Elapsed time: 1.4059202671051025 seconds
```



Comparação dos métodos (8 % dos dados para teste)

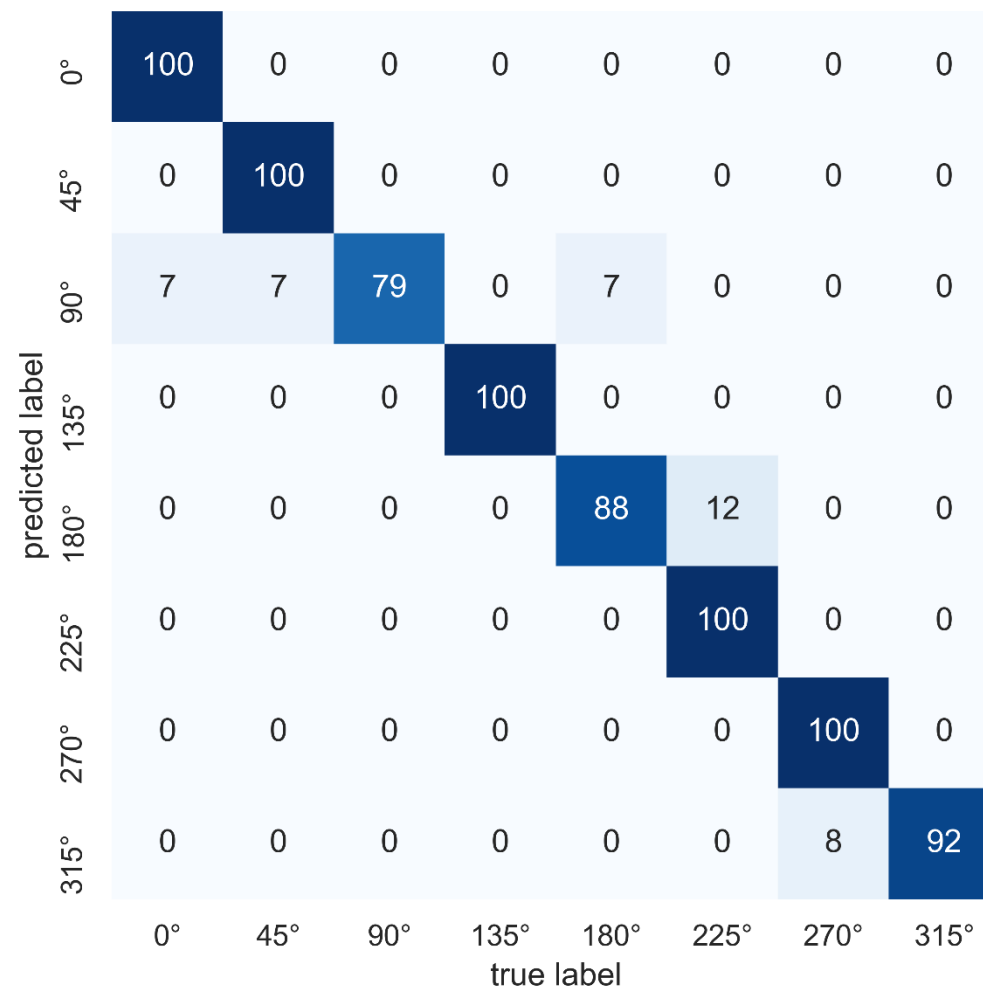
DNN com modificações

```
Epoch: 2000 cost= 0.164094241  
Optimization Finished!  
Elapsed time: 109.2904200553894 seconds  
Accuracy (train data): 0.9535674  
Accuracy (test data): 0.9350649
```



Bagging Classifier

```
Accuracy of model is: 93.5064935064935 %  
Elapsed time: 1.4059202671051025 seconds
```



Conclusões

- É proposto um novo método DF para ser usado em um sistema de vigilância por drones.
- O sistema compreende um receptor de canal único e uma antena direcional.
- Se obteve uma relação subjacente entre a potência média do sinal recebido no conjunto da n -ésima antena e θ .
- Os dados de treinamento e teste estavam predefinidos e sequencialmente distribuídos.
- Apenas 8% dos dados foram utilizados para validação.
- São propostas melhorias no modelo DNN e além é proposto um modelo com melhor desempenho.

Trabalhos futuros

- Sistema inteligente capaz de identificar o estado de drone de maneira precisa.
- Propor um sistema com um arreglo de antenas capaz de detectar o angulo de elevação, além do angulo azimuth.
- Um modelo de sistema com múltiplos canais.
- Aplicação das técnicas de múltiplo acesso não ortogonal para a detecção de drone.