NeuralNet

```
import numpy as np
import nltk
import random
```

```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
True
```

```
import json
```

```
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
```

```
def tokenize(sentence):
    """
    split sentence into array of words/tokens
    a token can be a word or punctuation character, or number
    """
    return nltk.word_tokenize(sentence)
```

```
def stem(word):
    """
    stemming = find the root form of the word
    examples:
    words = ["organize", "organizes", "organizing"]
    words = [stem(w) for w in words]
    -> ["organ", "organ", "organ"]
    """
    return stemmer.stem(word.lower())
```

```
def bag_of_words(tokenized_sentence, words):
    """
    return bag of words array:
    1 for each known word that exists in the sentence, 0 otherwise
    example:
    sentence = ["hello", "how", "are", "you"]
    words = ["hi", "hello", "I", "you", "bye", "thank", "cool"]
    bog   = [  0 ,    1 ,    0 ,   1 ,    0 ,    0 ,      0]
    """
    # stem each word
    sentence_words = [stem(word) for word in tokenized_sentence]
    # initialize bag with 0 for each word
    bag = np.zeros(len(words), dtype=np.float32)
    for idx, w in enumerate(words):
```

```
    for idx, w in enumerate(words):
        if w in sentence_words:
            bag[idx] = 1


    return bag
```

```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
```

```
with open("2-1.json", 'r') as f:
    intents = json.load(f)
```

```
print(intents)
```

```
{'intents': [{'Season': 'KHARIF', 'Sector': 'HORTICULTURE', 'Category': 'Fruits', 'Cr
```

```
ignore_words=['?','!','.',',','(',')','&','@']
```

```
all_words = []
tags = []
xy = []
patternize=[]
processed_patternize=[]
answer=[]
# loop through each sentence in our intents patterns
for intent in intents['intents']:
    tag = intent['QueryType']              # tag=intent eg-Fertilizer,market price,cultiva
    ans=intent['KccAns']
    bname=intent['BlockName']                  #answers for the query text
    answer.append(ans)
    # add to tag list
    tags.append(tag)
    pattern=intent['QueryText']           #querytext
    patternize.append(pattern)
        # tokenize each word in the sentence
    w = pattern.split(" ")
    w.append(bname)

        # add to our words list
    all_words.extend(w)
    i=w
    i = [stem(k) for k in i if k not in ignore_words]  # i) removing punctuation words fro
    i=" ".join(i)
    processed_patternize.append(i)
        # add to xy pair
    xy.append((w, tag))

y_train_1 = tags
```

```
print(xy)
```

```
[(['top', 'dressing', 'for', 'sapota', 'PALAYANKOTTAL'], 'Fertilizer Use and Availabi
```

```
print(processed_patternize)
```

```
['top dress for sapota palayankott', 'ask about weather report for tirupur avanashi',
```

```
print(all_words)
print(tags)
```

```
['top', 'dressing', 'for', 'sapota', 'PALAYANKOTTAL', 'Asking', 'about', 'Weather',
['Fertilizer Use and Availability', 'Weather', 'Weather', 'Weather', 'Market Informat
```

```
all_words = [stem(w) for w in all_words if w not in ignore_words]
# remove duplicates and sort
all_words = sorted(set(all_words))
tags = sorted(set(tags))
print(tags)
```

```
['Agriculture Mechanization', 'Animal Breeding', 'Animal Nutrition', 'Animal Producti
```

```
print(all_words)
```

```
['&brown', '(adt', '(ae),tiruvannamali', '(bio)', '(bpt)', '(chithiraipattam)', '(cow
```

```
remove_words=['(',')','&']          #  removing the symbols in (,),&
all_wordsn=[]
for i in all_words:
  if i[0] in remove_words or i[-1] in remove_words:
    if i[0] in remove_words:
      i=i[1:]
    if i[-1] in remove_words:
      i=i[:-1]
      all_wordsn.append(i)
  else:
    all_wordsn.append(i)
print(all_wordsn)
```

```
['bio', 'bpt', 'chithiraipattam', 'days', 'karthigaipattam', 'mn', 'n', 'navarai', 'c
```

```
print(len(tags))
```

```
37
```

```
all_words=all_wordsn
print(all_words)
```

```
['bio', 'bpt', 'chithiraipattam', 'days', 'karthigaipattam', 'mn', 'n', 'navarai', '
```

```
X_train = []
for (pattern_sentence, tag) in xy:
    # X: bag of words for each pattern_sentence
    bag = bag_of_words(pattern_sentence, all_words)     #all_words is a dictionary now.
    X_train.append(bag)
```

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(y_train_1)
list(le.classes_)
y_train = le.transform(y_train_1)
```

```
y_train
```

```
array([10, 35, 35, ..., 10,  7,  8])
```

```
X_train = np.array(X_train)
y_train = np.array(y_train)
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.33,rando
```

```
class ChatDataset(Dataset):

    def __init__(self):
        self.n_samples = len(X_train)
        self.x_data = X_train
        self.y_data = y_train

    # support indexing such that dataset[i] can be used to get i-th sample
    def __getitem__(self, index):
        return self.x_data[index], self.y_data[index]

    # we can call len(dataset) to return the size
    def __len__(self):
        return self.n_samples
```

```
dataset = ChatDataset()
```

```
print(len(dataset))
```

```
2160
```

```
batch_size=8
```

```
train_loader = DataLoader(dataset=dataset,batch_size=batch_size,shuffle=True,num_workers=0
```

```
class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.l1 = nn.Linear(input_size, hidden_size)
        self.l2 = nn.Linear(hidden_size, hidden_size)
        self.l3 = nn.Linear(hidden_size, num_classes)
        self.relu = nn.ReLU()
    def forward(self, x):
        out = self.l1(x)
        out = self.relu(out)
        out = self.l2(out)
        out = self.relu(out)
        out = self.l3(out)
        # no activation and no softmax at the end
        return out
```

```
# Hyper-parameters
num_epochs = 20
batch_size = 8
learning_rate = 0.001
input_size = len(X_train[0])
hidden_size = 8
output_size = len(tags)
print(input_size, output_size)
```

```
789 37
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
model = NeuralNet(input_size, hidden_size, output_size).to(device)
```

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

```
# Train the model
for epoch in range(num_epochs):
    for (words, labels) in train_loader:
        words = words.to(device)
        labels = labels.to(dtype=torch.long).to(device)
        # Forward pass
        outputs = model(words)
        # if y would be one-hot, we must apply
        # labels = torch.max(labels, 1)[1]
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        metrics="accuracy"


        print (f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

```
Epoch [20/20], Loss: 0.4217
Epoch [20/20], Loss: 1.0596
Epoch [20/20], Loss: 0.7678
Epoch [20/20], Loss: 2.1315
Epoch [20/20], Loss: 0.4241
Epoch [20/20], Loss: 1.2001
Epoch [20/20], Loss: 0.3336
Epoch [20/20], Loss: 0.1538
Epoch [20/20], Loss: 1.1641
Epoch [20/20], Loss: 0.5254
Epoch [20/20], Loss: 0.5941
Epoch [20/20], Loss: 0.5610
Epoch [20/20], Loss: 0.8309
Epoch [20/20], Loss: 0.8189
Epoch [20/20], Loss: 0.3554
Epoch [20/20], Loss: 0.5385
Epoch [20/20], Loss: 0.7004
Epoch [20/20], Loss: 0.2374
Epoch [20/20], Loss: 0.2934
Epoch [20/20], Loss: 2.6236
Epoch [20/20], Loss: 0.3230
Epoch [20/20], Loss: 0.3055
Epoch [20/20], Loss: 0.1838
Epoch [20/20], Loss: 1.1370
Epoch [20/20], Loss: 0.3526
Epoch [20/20], Loss: 0.1655
Epoch [20/20], Loss: 1.0128
Epoch [20/20], Loss: 1.0316
Epoch [20/20], Loss: 0.5125
Epoch [20/20], Loss: 0.2094
Epoch [20/20], Loss: 0.2425
Epoch [20/20], Loss: 0.4471
Epoch [20/20], Loss: 0.5757
Epoch [20/20], Loss: 0.4111
Epoch [20/20], Loss: 0.6593
Epoch [20/20], Loss: 0.5515
Epoch [20/20], Loss: 1.4713
Epoch [20/20], Loss: 0.6366
Epoch [20/20], Loss: 1.3512
Epoch [20/20], Loss: 0.8546
Epoch [20/20], Loss: 0.4394
Epoch [20/20], Loss: 1.1361
Epoch [20/20], Loss: 0.5419

Epoch [20/20], Loss: 0.9144
Epoch [20/20], Loss: 0.3834
Epoch [20/20], Loss: 0.3894
Epoch [20/20], Loss: 0.2930
Epoch [20/20], Loss: 0.4984
Epoch [20/20], Loss: 0.9996
Epoch [20/20], Loss: 0.1350
Epoch [20/20], Loss: 1.2912
Epoch [20/20], Loss: 0.6157
Epoch [20/20], Loss: 0.1198
Epoch [20/20], Loss: 0.3524
Epoch [20/20], Loss: 0.6564
Epoch [20/20], Loss: 0.8931
Epoch [20/20], Loss: 0.7720
Epoch [20/20], Loss: 0.6871
Epoch [20/20], Loss: 0.4968
Epoch [20/20], Loss: 0.4320
```

```
print(f'final loss: {loss.item():.4f}')
```

```
final loss: 0.4320
```

```
data = {
"model_state": model.state_dict(),
"input_size": input_size,
"hidden_size": hidden_size,
"output_size": output_size,
"all_words": all_words,
"tags": tags
}
```

```
FILE = "data.pth"
torch.save(data, FILE)
```

```
import pandas
```

```
print(f'training complete. file saved to {FILE}')
```

```
training complete. file saved to data.pth
```

```
import torch
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

```
data = torch.load(FILE)
```

```
res={}
for cl in range(0,len(patternize)):
  res.update({patternize[cl]:answer[cl]})
print(res)
```

```
{'top dressing for sapota': 'apply FYM 25kg+urea500gm+SSP500gm+potash750gm/tree once
```

```
input_size = data["input_size"]
hidden_size = data["hidden_size"]
output_size = data["output_size"]
all_words = data['all_words']
tags = data['tags']
model_state = data["model_state"]
```

```
model = NeuralNet(input_size, hidden_size, output_size).to(device)
```

```
model.load_state_dict(model_state)
```

```
<All keys matched successfully>
```

```
model.eval()
```

```
NeuralNet(
  (l1): Linear(in_features=789, out_features=8, bias=True)
  (l2): Linear(in_features=8, out_features=8, bias=True)
  (l3): Linear(in_features=8, out_features=37, bias=True)
  (relu): ReLU()
)
```

```
from difflib import get_close_matches
```

```
res={}
for cl in range(0,len(patternize)):
  res.update({patternize[cl]:answer[cl]})
print(res)
```

```
{'top dressing for sapota': 'apply FYM 25kg+urea500gm+SSP500gm+potash750gm/tree once
```

```
bot_name = "Sinegalatha"
print("Let's chat! (type 'quit' to exit)")
test=[]
while True:
    # sentence = "do you use credit cards?"
    sentencei = input("You: ")
    if sentencei == "quit":
        break
    sentence = sentencei.split(" ")
    X = bag_of_words(sentence, all_words)
    X = X.reshape(1, X.shape[0])
    X = torch.from_numpy(X).to(device)
    output = model(X)
    _, predicted = torch.max(output, dim=1)

    tag = tags[predicted.item()]
    print(tag)
    probs = torch.softmax(output, dim=1)
    prob = probs[0][predicted.item()]

    for intent in intents['intents']:
        if tag == intent["QueryType"]:
          test.append(intent["QueryText"])

    p=[]
    p=(get_close_matches(sentencei, test))
    if len(p)==0:
      print("Make a call to Kisan Call Centre ")
    else:
      u=res[p[0]]
      print(u)
```

```
Let's chat! (type 'quit' to exit)
You: paddy varieties
Varieties
Recommended for ADT 36, ADT 39, ASD 16, ASD 18, MDU 5, CO 47,CORH 3, ADT 43, ADT (R)
You: cow loan details
Government Schemes
Make a call to Kisan Call Centre
You: what is the weather report of thiruppur
Weather
recommended for having mostly cloudy weather condition
You: quit
```

```
predict_tag=[]
for X in X_test:
    X = X.reshape(1, X.shape[0])
    X = torch.from_numpy(X).to(device)
    output = model(X)
    _, predicted = torch.max(output, dim=1)

    print(predicted.item())
    predict_tag.append(predicted.item())
```

```
20
28
35
28
23
10
7
23
35
23
35
24
31
31
35
35
20
```

```
23
17
35
14
35
23
35
35
35
23
10
23
31
7
31
35
23
23
23
35
17
35
35
23
24
20
35
10
17
35
30
35
35
23
14
20
35
24
7
23
20
35
```

```
predict_train = np.array(predict_tag)
```

```
test_train = np.array(y_test)
```

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

```
a=accuracy_score(predict_train, test_train)
```

```
import random
random.seed(a)
```

```
print(a)
print(a)
```

```
0.7380281690140845
```

✓ 0s completed at 11:15 AM ● ×

Decision Tree

```
import numpy as np
import nltk
import random
```

```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
import json
```

```
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
```

```
def tokenize(sentence):
    """
    split sentence into array of words/tokens
    a token can be a word or punctuation character, or number
    """
    return nltk.word_tokenize(sentence)
```

```
def stem(word):
    """
    stemming = find the root form of the word
    examples:
    words = ["organize", "organizes", "organizing"]
    words = [stem(w) for w in words]
    -> ["organ", "organ", "organ"]
    """
    return stemmer.stem(word.lower())
```

```
def bag_of_words(tokenized_sentence, words):
    """
    return bag of words array:
    1 for each known word that exists in the sentence, 0 otherwise
    example:
    sentence = ["hello", "how", "are", "you"]
    words = ["hi", "hello", "I", "you", "bye", "thank", "cool"]
    bog   = [  0 ,    1 ,    0 ,   1 ,    0 ,    0 ,      0]
    """
    # stem each word
    sentence_words = [stem(word) for word in tokenized_sentence]
    # initialize bag with 0 for each word
    bag = np.zeros(len(words), dtype=np.float32)
    for idx, w in enumerate(words):
```

```
    for idx, w in enumerate(words):
        if w in sentence_words:
            bag[idx] = 1



    return bag
```

```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
```

```
with open('2-1.json', 'r') as f:
    intents = json.load(f)
```

```
print(intents)
```

```
{'intents': [{'Season': 'KHARIF', 'Sector': 'HORTICULTURE', 'Category': 'Fruits', 'Cr
```

```
ignore_words=['?','!','.',',','(',')','&','@']
```

```
all_words = []
tags = []
xy = []
patternize=[]
processed_patternize=[]
answer=[]
# loop through each sentence in our intents patterns
for intent in intents['intents']:
    tag = intent['QueryType']              # tag=intent eg-Fertilizer,market price,cultiva
    ans=intent['KccAns']
    bname=intent['BlockName']                  #answers for the query text
    answer.append(ans)
    # add to tag list
    tags.append(tag)
    pattern=intent['QueryText']           #querytext
    patternize.append(pattern)
        # tokenize each word in the sentence
    w = pattern.split(" ")
    w.append(bname)

        # add to our words list
    all_words.extend(w)
    i=w
    i = [stem(k) for k in i if k not in ignore_words]  # i) removing punctuation words fro
    i=" ".join(i)
    processed_patternize.append(i)
        # add to xy pair
    xy.append((w, tag))

y_train_1 = tags
```

```
print(xy)
```

```
[(['top', 'dressing', 'for', 'sapota', 'PALAYANKOTTAL'], 'Fertilizer Use and Availabi
```

```
print(processed_patternize)
```

```
['top dress for sapota palayankott', 'ask about weather report for tirupur avanashi',
```

```
print(all_words)
print(tags)
```

```
['top', 'dressing', 'for', 'sapota', 'PALAYANKOTTAL', 'Asking', 'about', 'Weather',
['Fertilizer Use and Availability', 'Weather', 'Weather', 'Weather', 'Market Informat
```

```
all_words = [stem(w) for w in all_words if w not in ignore_words]
# remove duplicates and sort
all_words = sorted(set(all_words))
tags = sorted(set(tags))
print(tags)
```

```
['Agriculture Mechanization', 'Animal Breeding', 'Animal Nutrition', 'Animal Producti
```

```
print(all_words)
```

```
['&brown', '(adt', '(ae),tiruvannamali', '(bio)', '(bpt)', '(chithiraipattam)', '(cow
```

```
remove_words=['(',')','&']          #  removing the symbols in (,),&
all_wordsn=[]
for i in all_words:
  if i[0] in remove_words or i[-1] in remove_words:
    if i[0] in remove_words:
      i=i[1:]
    if i[-1] in remove_words:
      i=i[:-1]
      all_wordsn.append(i)
  else:
    all_wordsn.append(i)
print(all_wordsn)
```

```
['bio', 'bpt', 'chithiraipattam', 'days', 'karthigaipattam', 'mn', 'n', 'navarai', 'c
```

```
print(len(tags))
```

```
37
```

```
all_words=all_wordsn
print(all_words)
```

```
['bio', 'bpt', 'chithiraipattam', 'days', 'karthigaipattam', 'mn', 'n', 'navarai', 'c
```

```
X_train = []
for (pattern_sentence, tag) in xy:
    # X: bag of words for each pattern_sentence
    bag = bag_of_words(pattern_sentence, all_words)     #all_words is a dictionary now.
    X_train.append(bag)
```

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(y_train_1)
list(le.classes_)
y_train = le.transform(y_train_1)
```

```
y_train
```

```
array([10, 35, 35, ..., 10,  7,  8])
```

```
X_train = np.array(X_train)
y_train = np.array(y_train)
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.33,rando
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
```

```
classifier.fit(X_train, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=0, splitter='best')
```

```
predi_tag= classifier.predict(X_test)
```

```
print(predi_tag)
```

```
[35 23 35 ... 23 24 20]
```

```
predi_train = np.array(predi_tag)
```

```
print(predi_train)
```

```
[35 23 35 ... 23 24 20]
```

```
testi_train = np.array(y_test)
```

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

```
a=accuracy_score(predi_train, testi_train)
```

```
import random
random.seed(a)
```

```
print(a)
```

```
0.723943661971831
```

RandomForest

```
import numpy as np
import nltk
import random
```

```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
import json
```

```
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
```

```
def tokenize(sentence):
    """
    split sentence into array of words/tokens
    a token can be a word or punctuation character, or number
    """
    return nltk.word_tokenize(sentence)
```

```
def stem(word):
    """
    stemming = find the root form of the word
    examples:
    words = ["organize", "organizes", "organizing"]
    words = [stem(w) for w in words]
    -> ["organ", "organ", "organ"]
    """
    return stemmer.stem(word.lower())
```

```
def bag_of_words(tokenized_sentence, words):
    """
    return bag of words array:
    1 for each known word that exists in the sentence, 0 otherwise
    example:
    sentence = ["hello", "how", "are", "you"]
    words = ["hi", "hello", "I", "you", "bye", "thank", "cool"]
    bog   = [  0 ,    1 ,    0 ,   1 ,    0 ,    0 ,      0]
    """
    # stem each word
    sentence_words = [stem(word) for word in tokenized_sentence]
    # initialize bag with 0 for each word
    bag = np.zeros(len(words), dtype=np.float32)
    for idx, w in enumerate(words):
```

```
    for idx, w in enumerate(words):
        if w in sentence_words:
            bag[idx] = 1



    return bag
```

```
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
```

```
with open('2-1.json', 'r') as f:
    intents = json.load(f)
```

```
print(intents)
```

```
{'intents': [{'Season': 'KHARIF', 'Sector': 'HORTICULTURE', 'Category': 'Fruits', 'Cr
```

```
ignore_words=['?','!','.',',','(',')','&','@']
```

```
all_words = []
tags = []
xy = []
patternize=[]
processed_patternize=[]
answer=[]
# loop through each sentence in our intents patterns
for intent in intents['intents']:
    tag = intent['QueryType']              # tag=intent eg-Fertilizer,market price,cultiva
    ans=intent['KccAns']
    bname=intent['BlockName']                  #answers for the query text
    answer.append(ans)
    # add to tag list
    tags.append(tag)
    pattern=intent['QueryText']           #querytext
    patternize.append(pattern)
        # tokenize each word in the sentence
    w = pattern.split(" ")
    w.append(bname)

        # add to our words list
    all_words.extend(w)
    i=w
    i = [stem(k) for k in i if k not in ignore_words]  # i) removing punctuation words fro
    i=" ".join(i)
    processed_patternize.append(i)
        # add to xy pair
    xy.append((w, tag))

y_train_1 = tags
```

```
print(xy)
```

```
[(['top', 'dressing', 'for', 'sapota', 'PALAYANKOTTAL'], 'Fertilizer Use and Availabi
```

```
print(processed_patternize)
```

```
['top dress for sapota palayankott', 'ask about weather report for tirupur avanashi',
```

```
print(all_words)
print(tags)
```

```
['top', 'dressing', 'for', 'sapota', 'PALAYANKOTTAL', 'Asking', 'about', 'Weather',
['Fertilizer Use and Availability', 'Weather', 'Weather', 'Weather', 'Market Informat
```

```
all_words = [stem(w) for w in all_words if w not in ignore_words]
# remove duplicates and sort
all_words = sorted(set(all_words))
tags = sorted(set(tags))
print(tags)
```

```
['Agriculture Mechanization', 'Animal Breeding', 'Animal Nutrition', 'Animal Producti
```

```
print(all_words)
```

```
['&brown', '(adt', '(ae),tiruvannamali', '(bio)', '(bpt)', '(chithiraipattam)', '(cow
```

```
remove_words=['(',')','&']          #  removing the symbols in (,),&
all_wordsn=[]
for i in all_words:
  if i[0] in remove_words or i[-1] in remove_words:
    if i[0] in remove_words:
      i=i[1:]
    if i[-1] in remove_words:
      i=i[:-1]
      all_wordsn.append(i)
  else:
    all_wordsn.append(i)
print(all_wordsn)
```

```
['bio', 'bpt', 'chithiraipattam', 'days', 'karthigaipattam', 'mn', 'n', 'navarai', '
```

```
print(len(tags))
```

```
37
```

```
all_words=all_wordsn
print(all_words)
```

```
['bio', 'bpt', 'chithiraipattam', 'days', 'karthigaipattam', 'mn', 'n', 'navarai', 'c
```

```
X_train = []
for (pattern_sentence, tag) in xy:
    # X: bag of words for each pattern_sentence
    bag = bag_of_words(pattern_sentence, all_words)     #all_words is a dictionary now.
    X_train.append(bag)
```

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
le.fit(y_train_1)
list(le.classes_)
y_train = le.transform(y_train_1)
```

```
y_train
```

```
array([10, 35, 35, ..., 10,  7,  8])
```

```
X_train = np.array(X_train)
y_train = np.array(y_train)
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.33,rando
```

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

```
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

```
predi_tag= classifier.predict(X_test)
```

```
print(predi_tag)
```

```
[35 23 35 ... 23 24 20]
```

```
predi_train = np.array(predi_tag)
```

Double-click (or enter) to edit

```
print(predi_train)
```

```
[35 23 35 ... 23 24 20]
```

```
testi_train = np.array(y_test)
```

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

```
a=accuracy_score(predi_train, testi_train)
```

```
import random
random.seed(a)
```

```
print(a)
```

```
0.7286384976525822
```

SEQUENTIAL

```
import tensorflow.keras
from tensorflow.keras.models import Sequential
#from tensorflow.python.keras.models import Sequential
from tensorflow.keras.layers import Dense
from·keras.utils·import·np_utils
from·tensorflow.keras.models·import·load_model
```

```
import pandas as pd
import numpy as np
import pickle as pk
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix
from sklearn.naive_bayes import GaussianNB
import re
from nltk.stem.porter import PorterStemmer
from sklearn.model_selection import train_test_split
#import keras
#from keras.models import Sequential
#from keras.layers import Dense
#from keras.utils import np_utils
#from keras.models import load_model
import tensorflow
```

```
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
```

```
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
#2.1 Get the file
downloaded = drive.CreateFile({'id':'1HyNRwhUyyr0qNuUB-bivHUOCruSbZoQw'}) # replace the id
downloaded.GetContentFile('intentsnew.csv')
```

```
#2.1 Get the file
downloaded = drive.CreateFile({'id':'1cIYL8hbis1S5SmOULTDzpBhz4JDp5tIj'}) # replace the id
downloaded.GetContentFile('now.csv')
```

```
#2.1 Get the file
downloaded = drive.CreateFile({'id':'134NT6Avkt8rH92nbke5zY0ZPDRYCMQnt'}) # replace the id
```

```
downloaded.GetContentFile('entity_model.sav')
```

```
#2.1 Get the file
downloaded = drive.CreateFile({'id':'1Nx6sriBNpFFm7c_8r02m9mSDsAmUUSym'}) # replace the id
downloaded.GetContentFile('EntityCountVectorizer.sav')
```

```
#2.1 Get the file
downloaded = drive.CreateFile({'id':'1OIrCd0kvFbGRtKdC95waKKigk6Bysxqg'}) # replace the id
downloaded.GetContentFile('intent_model.h5')
```

```
#2.1 Get the file
downloaded = drive.CreateFile({'id':'1YV5d5UE9B66Yul-rhf4epq1lM5oBwSFO'}) # replace the id
downloaded.GetContentFile('IntentCountVectorizer.sav')
```

```
#2.1 Get the file
downloaded = drive.CreateFile({'id':'145y4oTffcIQ7brxZwPPUoS8dNyXHak49'}) # replace the id
downloaded.GetContentFile('data-tags.csv')
```

```
#2.1 Get the file
downloaded = drive.CreateFile({'id':'1c8hmMc8NZY8r15Wf0xgG0AEnVAqtXqnT'}) # replace the id
downloaded.GetContentFile('test.csv')
```

```
#2.1 Get the file
downloaded = drive.CreateFile({'id':'1ILrnI9u33H1nW66nH3WuBFM5EzyKhdQb'}) # replace the id
downloaded.GetContentFile('2-1.json')
```

```
import json
with open('2-1.json', 'r') as f:
    intents = json.load(f)
```

```
patternize=[]
answer=[]
for intent in intents['intents']:
  pattern=intent['QueryText']
  patternize.append(pattern)
  ans=intent['KccAns']
  answer.append(ans)
```

```
from difflib import get_close_matches
```

```
res={}
for cl in range(0,len(patternize)):
  res.update({patternize[cl]:answer[cl]})
print(res)
```

```
{'top dressing for sapota': 'apply FYM 25kg+urea500gm+SSP500gm+potash750gm/tree once
```

```
    dataset = pd.read_csv('now.csv', names=["Query", "Intent","BlockName"])



    X = dataset["Query"]
    y = dataset["Intent"]
    z=dataset["BlockName"]
    print(z[1])
```

```
PALAYANKOTTAL
```

```
def trainIntentModel():
    # Load the dataset and prepare it to the train the model

    # Importing dataset and splitting into words and labels
    #dataset = pd.read_csv('datasets/intents.csv', names=["Query", "Intent"])
    dataset = pd.read_csv('now.csv', names=["Query", "Intent","BlockName"])



    X = dataset["Query"]
    y = dataset["Intent"]
    z=dataset["BlockName"]

    unique_intent_list = list(set(y))

    print("Intent Dataset successfully loaded!")

    # Clean and prepare the intents corpus
    queryCorpus = []
    ps = PorterStemmer()
    j=0
    for i in X:
      i=i+" "+str(z[j])
      j=j+1
      i = i.split(' ')
      tokenized_query = [ps.stem(word.lower()) for word in i]
      tokenized_query = ' '.join(tokenized_query)
      queryCorpus.append(tokenized_query)
    print(queryCorpus)
    countVectorizer= CountVectorizer(max_features=800)
    corpus = countVectorizer.fit_transform(queryCorpus).toarray()
    print(corpus.shape)
    print("Bag of words created!")

    # Save the CountVectorizer
    pk.dump(countVectorizer, open("IntentCountVectorizer.sav", 'wb'))
    print("Intent CountVectorizer saved!")

    # Encode the intent classes
    labelencoder_intent = LabelEncoder()
    y = labelencoder_intent.fit_transform(y)
    y = np_utils.to_categorical(y)
    print("Encoded the intent classes!")
    print(y)
    # Return a dictionary, mapping labels to their integer values
    res = {}
    for cl in labelencoder_intent.classes_:
        res.update({cl:labelencoder_intent.transform([cl])[0]})

    intent_label_map = res
    print(intent_label_map)
    print("Intent Label mapping obtained!")

    # Initialising the Aritifcial Neural Network
    classifier = Sequential()

    # Adding the input layer and the first hidden layer
    classifier.add(Dense(units = 96, kernel_initializer = 'uniform', activation = 'relu',

    # Adding the second hidden layer
    classifier.add(Dense(units = 96, kernel_initializer = 'uniform', activation = 'relu'))

    # Adding the output layer
    classifier.add(Dense(units = 38, kernel_initializer = 'uniform', activation = 'softmax

    # Compiling the ANN
    classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['

    # Fitting the ANN to the Training set
    classifier.fit(corpus, y, batch_size = 10, epochs = 100)

    return classifier, intent_label_map
```

```
from tensorflow.python.keras.optimizers import TFOptimizer
intent_model, intent_label_map = trainIntentModel()

# Save the Intent model
intent_model.save('intent_model.h5')

print("Intent model saved!")
```

```
Epoch 26/100
323/323 [==============================] - 1s 4ms/step - loss: 0.2099 - accuracy:
Epoch 27/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1994 - accuracy:
Epoch 28/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1934 - accuracy:
Epoch 29/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1865 - accuracy:

Epoch 30/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1872 - accuracy:
Epoch 31/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1797 - accuracy:
Epoch 32/100
Epoch 32/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1820 - accuracy:
Epoch 33/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1706 - accuracy:
Epoch 34/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1701 - accuracy:
Epoch 35/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1699 - accuracy:
Epoch 36/100
323/323 [==============================] - 1s 4ms/step - loss: 0.1684 - accuracy:
Epoch 37/100
323/323 [==============================] - 1s 4ms/step - loss: 0.1598 - accuracy:
Epoch 38/100
323/323 [==============================] - 1s 4ms/step - loss: 0.1572 - accuracy:
Epoch 39/100
323/323 [==============================] - 1s 4ms/step - loss: 0.1617 - accuracy:
Epoch 40/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1554 - accuracy:
Epoch 41/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1515 - accuracy:
Epoch 42/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1476 - accuracy:
Epoch 43/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1560 - accuracy:
Epoch 44/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1473 - accuracy:
Epoch 45/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1482 - accuracy:
Epoch 46/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1470 - accuracy:
Epoch 47/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1493 - accuracy:
Epoch 48/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1405 - accuracy:
Epoch 49/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1379 - accuracy:
Epoch 50/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1484 - accuracy:
Epoch 51/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1447 - accuracy:
Epoch 52/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1428 - accuracy:
Epoch 53/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1384 - accuracy:
Epoch 54/100
323/323 [==============================] - 1s 3ms/step - loss: 0.1380 - accuracy:
```

```
def trainEntityModel():
    # Importing dataset and splitting into words and labels
    dataset = pd.read_csv('data-tags.csv')

    X = dataset.iloc[:, :-1].values
    y = dataset.iloc[:, 1].values
#     X = X.reshape(630,)
    print(X)
    print("Entity Dataset successfully loaded!")

    entityCorpus=[]
    ps = PorterStemmer()
    # Stem words in X
    for word in X:
        if type(word[0]) is not str:
            word=str(word)
        word = [ps.stem(word[0])]
        entityCorpus.append(word)

    print(entityCorpus)
    X = entityCorpus
    from numpy import array
    X = array(X)
    X = X.reshape(628,)   # 542

    # Create a bag of words model for words
    from sklearn.feature_extraction.text import CountVectorizer
    cv = CountVectorizer(max_features=1500)
#     X = cv.fit_transform(X.astype('U')).toarray()
    X = cv.fit_transform(X).toarray()
    print("Entity Bag of words created!")

    # Save CountVectorizer state
    pk.dump(cv, open('EntityCountVectorizer.sav', 'wb'))
    print("Entity CountVectorizer state saved!")

    # Encoding categorical data of labels
    labelencoder_y = LabelEncoder()
    y = labelencoder_y.fit_transform(y.astype(str))
    print("Encoded the entity classes!")

    # Return a dict mapping labels to their integer values
    res = {}
    for cl in labelencoder_y.classes_:
        res.update({cl:labelencoder_y.transform([cl])[0]})
    entity_label_map = res
    print("Entity Label mapping obtained!")

    # Fit classifier to dataset
    classifier = GaussianNB()
    classifier.fit(X, y)
    print("Entity Model trained successfully!")

    # Save the entity classifier model
    pk.dump(classifier, open('entity_model.sav', 'wb'))
    print("Trained entity model saved!")

    return entity_label_map
```

```
# Load Entity model
entity_label_map = trainEntityModel()
```

```
      ['cashew']
      ['coconut']
      ['arecanuts']
      ['chilly']
```

```
['tobacco']
['soyabean']
['cardamom']
['?']
['pesticide']
['to']
['use']
['for']
['maize']
['.']
['fertilizer']
['for']
['ragi']
['field']
['?']
['required']
['rainfall']
['for']
['sugarcane']
['.']
['weather']
['this']
['week']
['!']
['my']
['name']
['is']
['ram']
['!']
['hi']
['i']
['am']
['vaishnavi']
['!']
['hey']
['hello']
['namaste']
['heya']
['wassup']

['?']
['cashew']
['coconut']
['arecanuts']
['chilly']
['tobacco']
['soyabean']
['cardamom']
['groundnut']
['gram']
['peas']
['green']
['chick']
['yellow']
['black']]
```

```
# ************************  Building Model is finished here ******************************
```

```
loadedEntityCV = pk.load(open('EntityCountVectorizer.sav', 'rb'))
loadedEntityClassifier = pk.load(open('entity_model.sav', 'rb'))
loadedEntityClassifier = pk.load(open('entity_model.sav', 'rb'))
```

```
def getEntities(query):
    query = loadedEntityCV.transform(query).toarray()

    response_tags = loadedEntityClassifier.predict(query)

    entity_list=[]
    for tag in response_tags:
        if tag in entity_label_map.values():
            entity_list.append(list(entity_label_map.keys())[list(entity_label_map.values(

    return entity_list
```

```
res={}
for cl in range(0,len(patternize)):
  res.update({patternize[cl]:answer[cl]})
print(res)
```

```
{'top dressing for sapota': 'apply FYM 25kg+urea500gm+SSP500gm+potash750gm/tree once
```

```
# Load model to predict user result
loadedIntentClassifier = load_model('intent_model.h5')
loaded_intent_CV = pk.load(open('IntentCountVectorizer.sav', 'rb'))

USER_INTENT = ""

while True:
    user_query = input()
    if user_query=='quit':
      break

    query = re.sub('[^a-zA-Z]', ' ', user_query)

    # Tokenize sentence
    query = query.split(' ')

    # Lemmatizing
    ps = PorterStemmer()
    tokenized_query = [ps.stem(word.lower()) for word in query]

    # Recreate the sentence from tokens
    processed_text = ' '.join(tokenized_query)

    # Transform the query using the CountVectorizer
    processed_text = loaded_intent_CV.transform([processed_text]).toarray()

    # Make the prediction
    predicted_Intent = loadedIntentClassifier.predict(processed_text)
#     print(predicted_Intent)
    result = np.argmax(predicted_Intent, axis=1)

    for key, value in intent_label_map.items():
```

```
    for key, value in intent_label_map.items():
        if value==result[0]:
            print(key)
            USER_INTENT = key
            break
    test=[]
    for intent in intents['intents']:
        if key == intent["QueryType"]:
          test.append(intent["QueryText"])

    p=[]
    p=(get_close_matches(user_query, test))
    if len(p)==0:
      print("Make a Call to Kisan Call Centre")
    else:
      u=res[p[0]]
      print(u)

    # Extract entities from text
    #entities = getEntities(tokenized_query)

    # Mapping between tokens and entity tags
    #token_entity_map = dict(zip(entities, tokenized_query))
    #print(token_entity_map)
```

```
weather report of thiruppur
WARNING:tensorflow:5 out of the last 294 calls to <function Model.make_predict_functi
Weather
recommended for having mostly cloudy weather condition
paddy varieties
Varieties
Recommended for ADT 36, ADT 39, ASD 16, ASD 18, MDU 5, CO 47,CORH 3, ADT 43, ADT (R)
quit
```

```
testset = pd.read_csv('test.csv', names=["QueryText", "QueryType"])


tx = testset["QueryText"]
ty = testset["QueryType"]
```

```
#2.1 Get the file
downloaded = drive.CreateFile({'id':'1M2mnGd-hnErAaPNL7nn10efiNAZdzl5q'}) # replace the id
downloaded.GetContentFile('test.json')
```

```
import json
#dataset = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/Intern-21-22/test.json', na
with open('test.json', 'r') as t:
    test = json.load(t)
```

```
test_t=[]
```

```
test_t=[]
type_t=[]
for i in test['intents']:
  q=i['QueryText']
  e=i['QueryType']

  type_t.append(e)

  query = re.sub('[^a-zA-Z]', ' ', q)

    # Tokenize sentence
  query = query.split(' ')

    # Lemmatizing
  ps = PorterStemmer()
  tokenized_query = [ps.stem(word.lower()) for word in query]

    # Recreate the sentence from tokens
  processed_text = ' '.join(tokenized_query)

    # Transform the query using the CountVectorizer
  processed_text = loaded_intent_CV.transform([processed_text]).toarray()

    # Make the prediction
  predicted_Intent = loadedIntentClassifier.predict(processed_text)
    #print(predicted_Intent)
  result = np.argmax(predicted_Intent, axis=1)

  for key, value in intent_label_map.items():
      if value==result[0]:
          print(key)
          USER_INTENT = key
          test_t.append(key)
          break
```

```
Market Information
Market Information
Market Information
Nutrient Management
Market Information
Market Information
Market Information
Market Information
Nutrient Management
Fertilizer Use and Availability
Nutrient Management
Nutrient Management
Market Information
Market Information
Plant Protection
Nutrient Management
Market Information
Nutrient Management
Market Information
Market Information
Fertilizer Use and Availability
Nutrient Management
Cultural Practices
```

```
Fertilizer Use and Availability
Nutrient Management
Fertilizer Use and Availability
Market Information
Nutrient Management
Nutrient Management
Nutrient Management
Plant Protection
Nutrient Management
Fertilizer Use and Availability
Fertilizer Use and Availability
Market Information
Market Information
Cultural Practices
Nutrient Management
Nutrient Management
Cultural Practices
Nutrient Management
Fertilizer Use and Availability
Market Information
Fertilizer Use and Availability
Fertilizer Use and Availability
Market Information
Nutrient Management
Nutrient Management
Nutrient Management
Nutrient Management
Nutrient Management
Nutrient Management
Nutrient Management
Market Information
Fertilizer Use and Availability
Fertilizer Use and Availability
Training and Exposure Visits
Market Information
Market Information
```

```
test_t = np.array(test_t)
type_train = np.array(type_t)
```

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

```
accuracy_score(test_t, type_t)
```

```
0.5416666666666666
```

```
import pandas as pd   #used to import amd manipulate data from various file formats such a
import numpy as np     #used for complex mathematical operations
import pickle as pk   #used for serializing and de-serializing Python object structures
from sklearn.feature_extraction.text import CountVectorizer  #Convert a collection of text
from sklearn.preprocessing import LabelEncoder   #Encode target labels with value between
from sklearn.naive_bayes import GaussianNB #used to update feature
import re       #used for computing regular expression
from nltk.stem.porter import PorterStemmer  #used to retrieve root word
from sklearn.model_selection import train_test_split  #used to train the model
import keras   #used to productize deep models
from keras.models import Sequential    #used to create models layer-by-layer.
from keras.layers import Dense   #used for changing the dimensions of the vector
```

```
dataset = pd.read_csv('2-1.csv')
```

```
queries = dataset["QueryText"]
intent = list(dataset["QueryType"])

print(queries)
print("Dataset successfully loaded!")
print(len(queries))
```

```
0                                 top dressing for sapota
1                 Asking about Weather report for Tirupur
2       Asking about Thiruppur district rainfall infor...
3                 Asking about Weather report for Tirupur
4                 Asking about Market rate for Ground nut
                              ...
3220      Asking about weather report for thiruvannamalai
3221     Asking about tapioca basal fertilizer management
3222             asking about banana fertilizer management
3223           asking about navarai season paddy varieties
3224                         asking about Cow loan details
Name: QueryText, Length: 3225, dtype: object
Dataset successfully loaded!
3225
0                                 top dressing for sapota
1                 Asking about Weather report for Tirupur
2       Asking about Thiruppur district rainfall infor...
3                 Asking about Weather report for Tirupur
4                 Asking about Market rate for Ground nut
                              ...
3220      Asking about weather report for thiruvannamalai
3221     Asking about tapioca basal fertilizer management
3222             asking about banana fertilizer management
3223           asking about navarai season paddy varieties
3224                         asking about Cow loan details
Name: QueryText, Length: 3225, dtype: object
Dataset successfully loaded!
3225
```

```
import json
```

```
with open('2-1.json', 'r') as f:
    intents = json.load(f)
```

```
ignore_words=['?','!','.',',',',','(',')','&','@']
```

```
def stem(word):
    """
    stemming = find the root form of the word
    examples:
    words = ["organize", "organizes", "organizing"]
    words = [stem(w) for w in words]
    -> ["organ", "organ", "organ"]
    """
    return stemmer.stem(word.lower())
```

```
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()
```

```
all_words = []
tags = []
xy = []
patternize=[]
processed_patternize=[]
answer=[]
# loop through each sentence in our intents patterns
for intent in intents['intents']:
    tag = intent['QueryType']              # tag=intent eg-Fertilizer,market price,cultiva
    ans=intent['KccAns']
    bname=intent['BlockName']                  #answers for the query text
    answer.append(ans)
    # add to tag list
    tags.append(tag)
    pattern=intent['QueryText']           #querytext
    patternize.append(pattern)
        # tokenize each word in the sentence
    w = pattern.split(" ")
    w.append(bname)

        # add to our words list
    all_words.extend(w)
    i=w
    i = [stem(k) for k in i if k not in ignore_words]  # i) removing punctuation words fro
    i=" ".join(i)
    processed_patternize.append(i)
        # add to xy pair
    xy.append((w, tag))

y_train_1 = tags
```

```
"""
#corpus creation of querydataset
queryCorpus = []
ps = PorterStemmer()
ps = PorterStemmer()

for query in queries:
    query = re.sub('[^a-zA-Z0-9]', ' ', query)

    # Tokenize sentence
    query = query.split(' ')

    # Lemmatizing
    tokenized_query = [ps.stem(word.lower()) for word in query]



    # Recreate the sentence from tokens
    tokenized_query = ' '.join(tokenized_query)



    # Add to corpus
    queryCorpus.append(tokenized_query)

    print(tokenized_query)
print(len(queryCorpus))
print("Corpus created")
"""
```

```
'\n#corpus creation of querydataset\nqueryCorpus = []\nps = PorterStemmer()\n\nfor qu
= re.sub(\'[^a-zA-Z0-9]\', \' \', query)\n\n    # Tokenize sentence\n    query = quer
matizing\n    tokenized_query = [ps.stem(word.lower()) for word in query]\n    \n
ence from tokens\n    tokenized_query = \' \'.join(tokenized_query)\n    \n\n\n    #
pus.append(tokenized_query)\n    \n    print(tokenized_query)\nprint(len(queryCorpus)
```

```
#feature vector generation
intent_CV= CountVectorizer(max_features=1500)  # convert a collection of text documents to
qcorpus = intent_CV.fit_transform(processed_patternize).toarray()  #performs fit and trans
print(qcorpus)
print(len(qcorpus))
print("Bag of words created!")
```

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
3225
Bag of words created!
```

```
# Encode the intents
labelencoder_intent = LabelEncoder()
intentlabel = labelencoder_intent.fit_transform(intent)
print("Encoded the classes!")
print(intent)
```

```
Encoded the classes!
```

```
['Fertilizer Use and Availability', 'Weather', 'Weather', 'Weather', 'Market Informat
```

```
# Splitting the dataset into the Training set and Test set
query_train, query_test, intent_train, intent_test = train_test_split(qcorpus, intentlabel

print("Dataset split into train and test set")
print(query_train)
print(intent_train)
```

```
Dataset split into train and test set
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
[23 10 35 ... 23 35 20]
```

```
# Fit the classifier to dataset
from sklearn.naive_bayes import GaussianNB   # used to update feature
classifier = GaussianNB()
classifier.fit(query_train, intent_train)
print("Model trained successfully!")
```

```
Model trained successfully!
```

```
intent_pred = classifier.predict(query_test)
print(len(query_test))
print(intent_pred)
```

```
807
[35 14 24 30 23 35 25  0  0 25 22 35 22  7 20 25 35 23 14 22 10 22  7  0
 25 22  4 17 20 23  8 36 25  9 22 31 35 22 22 22 22  0 22 10 35 35 23 23
 23 10 10 10 24 35  9 23 20 23 20 23 28 22 20  0 10 23 22 25  7 28 28 20
 20  7 17 25 28 28 22 22 23 10 20 22  7 28  8 16 35 14 22 17 35 22 22  0
 35 23  7 31  7 20 23 10 28 10  7 14 24 31 23 25 35 28  9 23 23 35 28 22
 24 23 22  2 17 17 10  7 28 35 23 11 10 17 22 22 20 31 35 23 17 22 28 35
 10 23 28 22 14 17  7 22 23 10 23  8 14 35 35 28 22  0  2 20 10 10 31 17
 28 22 20 22  7 25 17 14 28 31 35 22 17 23  7 25 20 10 22 17 23 20 28 35
 23 31 20 20 17  0 14 24 22 22 23 23  7 25 28 35 31 22 28 35 17 22  7 35
 10  0 22 35 22 14  7 22  0 23 28 22 23 22  0  0  0 22 30 35 22 28  7 31
 17 35 28 22 35 31 17 35 35 23 22 22 10 23 20 25 35 22 31 28 31 22  3 23
 22 25 28 35 31 35 17 23 35  0 35 23  0 23  9 22 22 23 22 10 23 28  7 10
 23 17 17  7 10 22 23 22 25 22 35 23  8 23 22 17 22 23 35 22 10 10 23 23
 31 35 22 23 35  7 20 20 23 23  7 23 22 10 10 20 22 24 35 20 28 10 35 10
 25 22 28 22 35 28 23 22 23 23 22  7 22 35 17 22  7  6 31 17 22 22  7  0
 22 28 22 17 15 17 14  7  0 22 22 35 22 25 35 22 35 25 22 10 17 22 35  7
 23 31  7 10 22 10 10  7  0 36 31 10 22 23 10 28 10 23 10 22 35 14 35 15
 22 10 24 10 23 23 22 20  7 17 20 28 22 28 25 22 20 35 24 22 17  0 22 17
 14 23 23 23 22 22 35  0 31 20 35 22 24 23  0  7 31 22  0 28 23 35 25 23
 22 10 22 35 31 28 10  7 20 10 23 25 10  0 28 31 20 22 22  0 22 10  0 14
 22 20 28 35 10 28 31 17 31 31 35 20  7 30 23 10 31 22 17 10 22 22 17  0
 31 35 36 22 20 35 28 23 28 14  7 28  7 23 23 14 35 22 17 22 22 22 22 20
 22  7  7 17 35 22 23 25 35  4  7 10 10 17  7 22 28 22 35  0 10  9 36 25
 28 28 31  7 25 20 10  0 22 25 35 23 35  7 22 23 23 23 22 11 22 10 10 17
 22 22 23 24 35 23  7  0 28 25 22 35 35 28 22 22 22 35 10 17 35 22 20 17
 28 17 17  7  0  7 31 22 10 22 28 10 25 35 22 17 23 22 22  7 35 22 23 17
 30 22 25 22 28 20 22 35 25 22 22  3 31 22 14 22 28 14 20 10 10 22 23 22
 23  7 23 20 25 28 22 22 24 22 31 25 31 28 23 10 10 22  7 20 25 23  7 20
 23 35 10 22 22 24 22  0 23  0 22 10 22  0 17 31 20 10 23 28 10 35  7 22
 23 10 25 22 22 35 20 10 22 22 22 28  3 22 35 22 22 28 20 28  7 23 33 10
 28 31  4 23 20 10 10 22 24 20 25 28 31 31 10 35 17  7  0 28 35  7  3 25
 22 28 28 10 20 22  0 22 23 10 28 31 22 20 23 10 22 31 22 14 23  0 35 17
 35 35 22 22  7 23 17 31 22 31 17 35 10 22 35 14  7 22 10 22 35 22 31 10
 22 31 22 10  7 24 22 35  7 10 10  0 10  7 22]
```

```
#importing confusion matrix   (while increasing the dataset 300 to 5000,the accuracy has i
from sklearn.metrics import confusion_matrix
confusion = confusion_matrix(intent_test, intent_pred)
print('Confusion Matrix\n')
print(confusion)

#importing accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('\nAccuracy: {:.2f}\n'.format(accuracy_score(intent_test, intent_pred)))

print('Micro Precision: {:.2f}'.format(precision_score(intent_test, intent_pred, average='
print('Micro Recall: {:.2f}'.format(recall_score(intent_test, intent_pred, average='micro'
print('Micro F1-score: {:.2f}\n'.format(f1_score(intent_test, intent_pred, average='micro'

print('Macro Precision: {:.2f}'.format(precision_score(intent_test, intent_pred, average='
print('Macro Recall: {:.2f}'.format(recall_score(intent_test, intent_pred, average='macro'
print('Macro F1-score: {:.2f}\n'.format(f1_score(intent_test, intent_pred, average='macro'

print('Weighted Precision: {:.2f}'.format(precision_score(intent_test, intent_pred, averag
print('Weighted Recall: {:.2f}'.format(recall_score(intent_test, intent_pred, average='wei
print('Weighted F1-score: {:.2f}'.format(f1_score(intent_test, intent_pred, average='weigh
```

```
Confusion Matrix

[[ 0  0  0 ...  0  0  0]
 [ 0  0  0 ...  0  0  0]
 [ 0  0  0 ...  0  0  0]
 ...
 [ 0  0  0 ...  0  0  0]
 [ 1  0  0 ...  0 64  0]
 [ 0  0  0 ...  0  0  3]]

Accuracy: 0.35

Micro Precision: 0.35
Micro Recall: 0.35
Micro F1-score: 0.35

Macro Precision: 0.29
Macro Recall: 0.31
Macro F1-score: 0.26

Weighted Precision: 0.56
Weighted Recall: 0.35
Weighted F1-score: 0.40
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: Undef
  _warn_prf(average, modifier, msg_start, len(result))
```