**档案**

类型

名称

最新提交消息

提交时间

[演示](https://github.com/codingforentrepreneurs/The-Hello-World-of-Machine-Learning/tree/master/demo)

[2-演示](https://github.com/codingforentrepreneurs/The-Hello-World-of-Machine-Learning/commit/4933b278849b1fc1281ffd72728a1ef78389ed21)

上个月

[笔记本](https://github.com/codingforentrepreneurs/The-Hello-World-of-Machine-Learning/tree/master/notebooks)

[项目提交](https://github.com/codingforentrepreneurs/The-Hello-World-of-Machine-Learning/commit/0e4d4977f72986d1ad766ef94219bead55d48e9e)

上个月

[.gitignore](https://github.com/codingforentrepreneurs/The-Hello-World-of-Machine-Learning/blob/master/.gitignore)

[项目提交](https://github.com/codingforentrepreneurs/The-Hello-World-of-Machine-Learning/commit/0e4d4977f72986d1ad766ef94219bead55d48e9e)

上个月

[点文件](https://github.com/codingforentrepreneurs/The-Hello-World-of-Machine-Learning/blob/master/Pipfile)

[项目提交](https://github.com/codingforentrepreneurs/The-Hello-World-of-Machine-Learning/commit/0e4d4977f72986d1ad766ef94219bead55d48e9e)

上个月

[Pipfile.lock](https://github.com/codingforentrepreneurs/The-Hello-World-of-Machine-Learning/blob/master/Pipfile.lock)

[项目提交](https://github.com/codingforentrepreneurs/The-Hello-World-of-Machine-Learning/commit/0e4d4977f72986d1ad766ef94219bead55d48e9e)

上个月

[自述文件](https://github.com/codingforentrepreneurs/The-Hello-World-of-Machine-Learning/blob/master/README.md)

[添加了自述文件](https://github.com/codingforentrepreneurs/The-Hello-World-of-Machine-Learning/commit/c8e0189dc52f394b9efdaafd124eb53865c2752d)

上个月

[ml-hello-world.code-workspace](https://github.com/codingforentrepreneurs/The-Hello-World-of-Machine-Learning/blob/master/ml-hello-world.code-workspace)

[项目提交](https://github.com/codingforentrepreneurs/The-Hello-World-of-Machine-Learning/commit/0e4d4977f72986d1ad766ef94219bead55d48e9e)

上个月

**自述文件**

机器学习只是从数据中学习计算机，而不是遵循配方。它的目的是模仿人们（也许还有其他动物）如何学习，同时仍然以数学为基础。

这篇文章旨在使您开始使用基本的机器学习模型。

一个聊天机器人。

现在，我们不会重新创建Alexa，Siri，Cortana或Google Assistant，但我们将从头开始创建一个全新的机器学习程序。

假设您了解一些Python编程，那么本教程将非常容易。

观看与此帖子相对应的[整个系列](https://github.com/codingforentrepreneurs/The-Hello-World-of-Machine-Learning/blob/master/projects/hello-world-machine-learning)。

**步骤1：我们的数据是什么？**

机器学习需要数据来实际学习。机器还没有像您和我一样学习，但是机器通过在似乎对您和我来说并不明显的事物中找到模式来学习，我们将在整篇文章中看到很多内容。

在定义数据之前，让我们谈谈这个ML（机器学习）项目的目标：

用预先定义的答案回答一些“随机”问题。

这是我们将尝试解决的问题：

**场景1**

法案： Hi there, what time do you open tomorrow for lunch?

机器人： Our hours are 9am-10pm everyday.

**方案2**

卡伦： Can I speak to your manager?

机器人： You can contact our customer support at 555-555-555.5

**场景3**

韦德： What type of products do you have?

机器人： We carry various food items including tacos, nachos, burritos, and salads.

让我们将其放入python格式：

对话 = [

{

“ customer”：“嗨，你明天几点几点吃午饭？” ，

“响应”：“我们的时间是每天上午9点至晚上10点。”

}，

{

“客户”：“我可以和您的经理谈谈吗？” ，

“响应”：“您可以通过555-555-5555与我们的客户支持联系。”

}，

{

“ customer”：“您拥有什么类型的产品？” ，

“响应”：“我们运载各种食品，包括炸玉米饼，玉米片，墨西哥卷饼和沙拉。”

}

]

如果没有机器学习，我们的机器人将如下所示（取消注释要运行的下一个单元）：

＃while

True：＃my\_input = input（“您的问题是什么？\ n”）＃response = None

＃对于对话中的

convo：＃if convo ['customer'] == my\_input：＃response = convo

['response']

＃如果响应！=无：

＃打印（响应）

＃中断

＃打印（“我不知道”）

＃继续

立即，您应该会看到此食谱中的巨大缺陷；如果客户没有以特定的预定义方式提出问题，则机器人会失败并最终导致失败。

举几个例子：-如果客户说什么*时候开门呢？*您已经知道响应是什么？-如果顾客说，*您卖汉堡吗？* -如果客户说：*如何通过电话与您联系？*

我敢肯定，您可以拿出更多更多关于此事真正消失的例子。

因此，让我们通过添加tags描述初始问题的内容来进一步清理对话数据。

convos\_one = [

{

“ customer”：“嗨，你明天几点几点吃午饭？” ，

“标签”：[ “开放”，“关闭”，“营业时间” ]，

}，

{

“客户”：“我可以和您的经理谈谈吗？” ，

“ tags”：[ “ customer\_support” ]，

}，

{

“客户”：“食物很棒，谢谢！” ，

“ tags”：[ “ customer\_support”，“ feedback” ]，

}，

{

“ customer”：“您拥有什么类型的产品？” ，

“标签”：[ “产品”，“菜单”，“库存”，“食品” ]，

}

]

convos\_two = [

{

“ customer”：“您的厨房开多久了？” ，

“标签”：[ “打开”，“小时”，“关闭” ]，

}，

{

“ customer”：“我的订单准备不正确，如何解决此问题？” ，

“ tags”：[ “ customer\_support” ]，

}，

{

“ customer”：“您吃哪种肉？” ，

“标签”：[ “菜单”，“产品”，“库存”，“食品” ]，

}

]

convos\_three = [

{

“ customer”：“您的餐厅什么时候开放？” ，

“标签”：[ “开放时间”，“小时数” ]，

}，

{

“ customer”：“您什么时候开放吃晚餐？” ，

“标签”：[ '开口'，'小时'，“关闭” ]，

}，

{

“客户”：“我如何与您联系？” ，

“标签”：[ “联系人”，“客户支持” ]

}

]

您看到这里正在发生趋势吗？对于餐厅机器人来说，提出各种问题真的很容易。还很容易看出，尝试和硬编码条件来处理客户可能遇到的所有类型的查询将是多么具有挑战性。

我确定您已经听说过需要大量数据才能进行机器学习。我将添加一点，您需要大量数据才能拥有*令人敬畏的*机器学习项目。一个为街上一家夫妻商店的简单机器人，现在还不需要*令人敬畏的方式*。他们需要简单，平易近人，易于解释。就是这样。这不是*数百万*行数据点的黑匣子。就像是在现场标记了20个问题。

在许多方面，当今（2020年代）的机器学习就像1990年代的互联网一样。人们已经听说过它并“得到了它”，并且感觉这仅仅是超级书呆子才知道该怎么做的神奇magic头。哈。超级书呆子。

现在我们有了起始数据，让我们为机器学习做准备。

首先，让我们合并所有对话：

数据集 = convos\_one + convos\_two + convos\_three

数据集

[{'customer': 'Hi there, what time do you open tomorrow for lunch?',

'tags': ['opening', 'closing', 'hours']},

{'customer': 'Can I speak to your manager?', 'tags': ['customer\_support']},

{'customer': 'The food was amazing thank you!',

'tags': ['customer\_support', 'feedback']},

{'customer': 'What type of products do you have?',

'tags': ['products', 'menu', 'inventory', 'food']},

{'customer': 'How late is your kitchen open?',

'tags': ['opening', 'hours', 'closing']},

{'customer': 'My order was prepared incorrectly, how can I get this fixed?',

'tags': ['customer\_support']},

{'customer': 'What kind of meats do you have?',

'tags': ['menu', 'products', 'inventory', 'food']},

{'customer': 'When does your dining room open?',

'tags': ['opening', 'hours']},

{'customer': 'When do you open for dinner?',

'tags': ['opening', 'hours', 'closing']},

{'customer': 'How do I contact you?',

'tags': ['contact', 'customer\_support']}]

我们的对话有关键customer和关键tags。这些是该项目的任意名称，您可以随意更改它们。只要记住，customer平等input和tags平等output。这是有道理的，因为在将来，我们需要随机的客户输入，例如What's the menu specials today和预测的标签输出类似menu或类似的东西。

机器学习有各种各样的术语和首字母缩写词，这常常使它有些混乱。通常，只要记住您有inputs一些目标outputs。这就是我的意思：

* customer：这些值实际上是input我们的ML项目的值。输入值有时被称为source，feature，training，X，X\_train/ X\_test/ X\_valid，和其他几个人。
* tags：这些值实际上是output我们的ML项目的值。输出值有时被称为target，labels，y，y\_train/ y\_test/ y\_valid，classes/ class，和其他几个人。

我们正在使用一种称为的机器学习技术，supervised learning这意味着我们同时提供inputs和outputs模型。这两个数据点都是我们想到的已知数据。如您所知，tags（或labels/ outputs）由人（即您和我）决定，但最终可以由ML模型本身决定，然后由人验证。这样做会使模型变得越来越好。还有许多其他技术，但是supervised learning对于初学者来说，这是最通行的。

**准备ML**

现在我们有了数据，是时候将其放入一种适用于计算机的格式了。如您所知，计算机在数字方面很出色，而文本却不那么出色。在这种情况下，我们必须将文本转换为数字。

使用[scikit-learn](https://scikit-learn.org/stable/index.html)库可以使此过程变得简单。因此，让我们通过取消注释该单元格在下面安装它。

＃！pip install scikit-learn

首先，让我们将customer和tag数据转换为2个单独的列表，其中每个项目的索引都与另一个项目的索引相对应。

X = [customer\_convo\_1, customer\_convo\_2, ...]

y = [convo\_1\_tags, convo\_2\_tags, ...]

这是非常标准的做法，这样X[0]是input对应于y[0] output，X[1]是input对应于y[1] output等等。

输入 = [ x [ 'customer' ] for data in data ]

print（输入）

['Hi there, what time do you open tomorrow for lunch?', 'Can I speak to your manager?', 'The food was amazing thank you!', 'What type of products do you have?', 'How late is your kitchen open?', 'My order was prepared incorrectly, how can I get this fixed?', 'What kind of meats do you have?', 'When does your dining room open?', 'When do you open for dinner?', 'How do I contact you?']

输出 = [ X [ '标签' ] 为 X 在 数据集中 ]

打印（输出）

[['opening', 'closing', 'hours'], ['customer\_support'], ['customer\_support', 'feedback'], ['products', 'menu', 'inventory', 'food'], ['opening', 'hours', 'closing'], ['customer\_support'], ['menu', 'products', 'inventory', 'food'], ['opening', 'hours'], ['opening', 'hours', 'closing'], ['contact', 'customer\_support']]

断言（len（输入）== len（输出））

如果您具有AssertionError上述条件，则意味着您inputs与outputs并不平衡。检查您的数据源，以确保每个数据源input都有相应的output值。

让我们验证一下这些数据的位置，以显示我们实际更改了多少数据：

idx = 4

打印（输入 [ idx ]，输出 [ idx ]）

打印（数据集 [ idx ]）

How late is your kitchen open? ['opening', 'hours', 'closing']

{'customer': 'How late is your kitchen open?', 'tags': ['opening', 'hours', 'closing']}

**预测功能**

机器学习的目标是产生一个接受inputs并产生outputs（预测）的函数。以下是概念上的表示：

def my\_pred\_function（输入）：

＃pred

输出 = 输入 \* 0.39013 ＃此小数表示模型将执行的操作。

返回 输出

现在我们需要将每个inputs列表和outputs列表转换成矩阵，以便我们的机器学习可以进行机器学习。

scikit-learn有一个简单的方法可以做到这一点。首先，让我们关注最简单的inputs（即customer对话）。

来自 sklearn。feature\_extraction。文本 导入 CountVectorizer

vectorizer = CountVectorizer（）

X = 矢量化器。fit\_transform（输入）

技术说明：scikit-learn将我们的数据转换为一维矩阵的集合。我们需要使用矩阵，以便可以进行矩阵乘法（这就是机器学习的原理）。在numpy说话，X是array的array秒。如果要查看创建的实际矢量，请签出X.toarray()并看到。

X。形状

(10, 43)

X.shape 对描述我们的数据很有用。

X.shape[0]是指我们final\_convos列表中的对话次数。所以，X.shape[0] == len(final\_convos)和X.shape[0] == len(inputs)

X.shape[1]指words我们的数据数。在CountVectorizer我们这样做。机器学习一词features与我们的数据有关。您可以通过以下方式查看所有features（words减去标点符号）：

词 = 向量器。get\_feature\_names（）

打印（单词）

['amazing', 'can', 'contact', 'dining', 'dinner', 'do', 'does', 'fixed', 'food', 'for', 'get', 'have', 'hi', 'how', 'incorrectly', 'is', 'kind', 'kitchen', 'late', 'lunch', 'manager', 'meats', 'my', 'of', 'open', 'order', 'prepared', 'products', 'room', 'speak', 'thank', 'the', 'there', 'this', 'time', 'to', 'tomorrow', 'type', 'was', 'what', 'when', 'you', 'your']

如您所见，矢量化器的词汇量非常有限。自然地，这意味着我们的机器学习项目将*总是会*误解一些关键的对话，这没关系。我们项目的目标是用它来得到它的第一份工作，让客户（或自己），所以我们可以*提高*用新的数据它的时候了（因此再提高它）。

len（字）

43

**准备输出（labels）**

我们的每个输入都有一个标签列表，而不仅仅是一个标签。让我们看看我的意思：

打印（输入 [ 0 ]，输出 [ 0 ]，实例（输出 [ 0 ]，列表））

Hi there, what time do you open tomorrow for lunch? ['opening', 'closing', 'hours'] True

在机器学习中，这意味着multi-label分类，因为output每个input值都有多个可能的值。这是一个比single标签更具挑战性的问题，但对于聊天机器人项目绝对是必需的。

单个标签数据集如下所示：

Input: Hi there, how are you doing today?

Output: not\_spam

Input: Free CELL phones just text 3ED#2

Output: spam

注意输出是一个单一str而不是一个list的str值。如果继续沿着这条道路前进，我们的数据将*始终*分为两类：spam或not\_spam。这种类型的分类称为binary分类，因为只有两种可能的预测类别。

在我们的项目中，我们的input值*可以*分为多个class项目1 class或完全没有class项目。（请记住，class= output tag= label）

来自 sklearn。预处理 导入 MultiLabelBinarizer

mlb = MultiLabelBinarizer（）

y = mlb。fit\_transform（输出）

您可能会考虑像训练数据（aka ）一样运行fit\_transform，但这不适用于多标签分类。为此，我们需要。CountVectorizerinputsMultiLabelBinarizer

美国职棒大联盟。classes\_

array(['closing', 'contact', 'customer\_support', 'feedback', 'food',

'hours', 'inventory', 'menu', 'opening', 'products'], dtype=object)

调用mlb.classes\_为我们提供了如何在中定义类的确切顺序y。因此y[0]对应于，outputs[0]但以数字代替单词。它太酷了。要从技术上查看此代码，请运行以下代码：

print(y[0])

# map to classes with `zip`

y0\_mapped\_to\_classes = dict(zip(mlb.classes\_, y[0]))

print(y0\_mapped\_to\_classes)

然后比较：

sorted(outputs[0]) == sorted([k for k,v in y0\_mapped\_to\_classes.items() if v == 1])

ÿ

array([[1, 0, 0, 0, 0, 1, 0, 0, 1, 0],

[0, 0, 1, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 1, 1, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 1, 1, 0, 1],

[1, 0, 0, 0, 0, 1, 0, 0, 1, 0],

[0, 0, 1, 0, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 1, 1, 0, 1],

[0, 0, 0, 0, 0, 1, 0, 0, 1, 0],

[1, 0, 0, 0, 0, 1, 0, 0, 1, 0],

[0, 1, 1, 0, 0, 0, 0, 0, 0, 0]])

在这里，我们可以看到使用sklearn's为我们生成的矩阵MultiLabelBinarizer。这是一个单数组的数组。

**one-hot**是一个术语，指的是我们用于此特定模型的编码类型。这是机器学习中非常普遍的做法。本质上将数据转换为1s和0s，而不是字符串或任何其他数据类型。

ÿ。形状

(10, 10)

y.shape 以类似的方式来描述我们的数据很有用 X.shape

y.shape[0]是指我们final\_convos列表中的对话次数。因此，y.shape[0] == len(final\_convos)与y.shape[0] == len(outputs)和y.shape[0] == X.shape[0]

y.shape[1]指tags每个对话具有的所有可能值的唯一值；永远不会重复使用MultiLabelBinarizer。

断言 y。形状 [ 0 ] == X。形状 [ 0 ]

断言 y。形状 [ 0 ] == len（输入）

如果您在AssertionError此处看到，则assert len(inputs) == len(outputs)与上面的错误完全相同。您的数据不平衡。

**培训 scikit-lean**

来自 sklearn。多输出 进口 MultiOutputClassifier

从 sklearn。合奏 进口 RandomForestClassifier

森林 = RandomForestClassifier（random\_state = 1）

模型 = MultiOutputClassifier（森林，n\_jobs = - 1）

模型。适合（X，y）

MultiOutputClassifier(estimator=RandomForestClassifier(random\_state=1),

n\_jobs=-1)

**预测**

txt = “嗨，您什么时候关闭？”

input\_vector = 矢量化器。转换（[ txt ]）

input\_vector

<1x43 sparse matrix of type '<class 'numpy.int64'>'

with 4 stored elements in Compressed Sparse Row format>

output\_vector = 模型。预测（input\_vector）

打印（output\_vector）

[[0 0 0 0 0 0 0 0 0 0]]

preds = {}

类 = MLB。classes\_

用于 我，VAL 在 枚举（output\_vector [ 0 ]）：

preds [ 类 [ 我 ]] = VAL

Preds

{'closing': 0,

'contact': 0,

'customer\_support': 0,

'feedback': 0,

'food': 0,

'hours': 0,

'inventory': 0,

'menu': 0,

'opening': 0,

'products': 0}

def label\_predictor（txt = 'Hello world'）：

＃pred

input\_vector = vectorizer。转换（[ txt ]）

output\_vector = 模型。预测（input\_vector）

preds = {}

类 = MLB。classes\_

用于 我，VAL 在 枚举（output\_vector [ 0 ]）：

preds [ 类 [i ]] = val

return preds

label\_predictor（）

{'closing': 0,

'contact': 0,

'customer\_support': 0,

'feedback': 0,

'food': 0,

'hours': 0,

'inventory': 0,

'menu': 0,

'opening': 0,

'products': 0}

label\_predictor（“什么时候打开？”）

{'closing': 0,

'contact': 0,

'customer\_support': 0,

'feedback': 0,

'food': 0,

'hours': 1,

'inventory': 0,

'menu': 0,

'opening': 1,

'products': 0}

label\_predictor（“明天什么时候营业？”）

{'closing': 0,

'contact': 0,

'customer\_support': 0,

'feedback': 0,

'food': 0,

'hours': 0,

'inventory': 0,

'menu': 0,

'opening': 0,

'products': 0}

**导出模型以供重新使用**

导入 泡菜

＃类

＃模型

＃向量器

model\_data = {

“类”：列表（MLB。classes\_），

“ 模式”：模型，

“矢量化”：向量化

}

与 开放式（“model.pkl”，“世行”）作为 ˚F：

咸菜。转储（model\_data，f）

**重用导出模型**

model\_loaded\_data = {}

与 开放（“model.pkl” ，'RB' ）作为 ˚F：

model\_loaded\_data = 泡菜。负载（˚F。阅读（））

def label\_predictor\_from\_export（txt = 'Hello world'，

vectorizer = None，

model = None，

classes = []，

\* args，

\*\* kwargs）：

＃pred

assert（vectorizer ！= None）

assert（model ！= None）

input\_vector = 向量器。转换（[ txt ]）

output\_vector = 模型。预测（input\_vector）

断言（LEN（output\_vector [ 0 ]）== LEN（类））

preds = {}

类 = MLB。classes\_

用于 我，VAL 在 枚举（output\_vector [ 0 ]）：

preds [ 类[ i ]] = val

return preds

label\_predictor\_from\_export（“厨房什么时候关闭？”，\*\* model\_loaded\_data）

{'closing': 0,

'contact': 0,

'customer\_support': 0,

'feedback': 0,

'food': 0,

'hours': 1,

'inventory': 0,

'menu': 0,

'opening': 1,

'products': 0}

**重新训练新数据**

来自 sklearn。feature\_extraction。文本 导入 CountVectorizer

从 sklearn。预处理 从sklearn 导入 MultiLabelBinarizer

。多输出进口MultiOutputClassifier 从sklearn。集成导入RandomForestClassifier

DEF 培养（数据集，train\_col = '客户'，label\_col = '标签'，export\_path = 'model.pkl' ）：

输入 = [ X [ train\_col ] 对于 X 在 数据集中 ]

输出 = [ X [ label\_col ] 对于 X 在 数据集 ]

断言（len（输入）== len（输出））

vectorizer = CountVectorizer（）

X = vectorizer。fit\_transform（输入）

MLB = MultiLabelBinarizer（）

ý = MLB。fit\_transform（输出）

类 = 列表（MLB。classes\_）

森林 = RandomForestClassifier（random\_state = 1）

模型 = MultiOutputClassifier（森林，n\_jobs = - 1）

模型。配合（X，ÿ）

model\_data = {

“类”：列表（MLB。classes\_），

“ 模式”：模型，

“矢量化”：向量化

}

与 开放（export\_path，把wb）作为 ˚F：

泡菜。转储（model\_data，f）

返回 export\_path

＃数据集

火车（数据集，export\_path = 'model2.pkl'）

'model2.pkl'

model\_loaded\_data = {}

与 开放（“model2.pkl” ，'RB' ）作为 ˚F：

model\_loaded\_data = 泡菜。负载（˚F。阅读（））

label\_predictor\_from\_export（“您最喜欢的菜单项是什么？”，\*\* model\_loaded\_data）

{'closing': 0,

'contact': 0,

'customer\_support': 0,

'feedback': 0,

'food': 0,

'hours': 0,

'inventory': 0,

'menu': 0,

'opening': 0,

'products': 0}

**用熊猫存储数据集**

！p ip 安装 熊猫

Requirement already satisfied: pandas in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (1.0.5)

Requirement already satisfied: numpy>=1.13.3 in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (from pandas) (1.19.1)

Requirement already satisfied: pytz>=2017.2 in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (from pandas) (2020.1)

Requirement already satisfied: python-dateutil>=2.6.1 in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (from pandas) (2.8.1)

Requirement already satisfied: six>=1.5 in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (from python-dateutil>=2.6.1->pandas) (1.15.0)

将 熊猫 作为 pd 导入

df = pd。DataFrame（数据集）

df。头（n = 100）

<style scoped> .dataframe tbody tr th：only-of-type {vertical-align：middle; }

.dataframe tbody tr th {

vertical-align: top;

}

.dataframe thead th {

text-align: right;

}

</ style>

|  | **顾客** | **标签** |
| --- | --- | --- |
| **0** | 嗨，你明天几点钟开始... | [营业时间，营业时间] |
| **1个** | 我可以和你的经理谈谈吗？ | [客户支持] |
| **2** | 食物很棒，谢谢！ | [客户支持，反馈] |
| **3** | 您有什么类型的产品？ | [产品，菜单，库存，食品] |
| **4** | 您的厨房开放多久？ | [营业时间，营业时间，营业时间] |
| **5** | 我的订单准备不正确，我该如何... | [客户支持] |
| **6** | 你吃哪种肉？ | [菜单，产品，库存，食品] |
| **7** | 您的餐厅什么时候开放？ | [营业时间] |
| **8** | 您什么时候开晚餐？ | [营业时间，营业时间，营业时间] |
| **9** | 我该如何联系您？ | [联系人，customer\_support] |

df。to\_pickle（“ dataset.pkl”）

＃df = pd.read\_pickle（“ dataset.pkl”）

＃og\_df.head（）

＃og\_df.iloc [0] ['tags'] [0]

new\_dataset = df。to\_dict（“ records”）

＃print（new\_dataset）

**添加到数据集**

＃df = df.append（{“ customer”：“谁是经理？”，“ tags”：[“ customer\_support”]}，ignore\_index = True）

＃df.head（n = 100）

def append\_to\_df（df）：

df = df。副本（）

，而 真：

customer\_input = 输入（“？问题是什么\ n ”）

tags\_input = 输入（“？标签用逗号分隔\ n ”）

如果 tags\_input ！= 无：

tags\_input = tags\_input。如果不是，请分割（“，”）

isinstance（标签输入，列表）：

如果客户输入= 无且标签输入！= 无：

df = df ，则标签输入 = [ 标签输入 ]

。追加（{ “ customer”：customer\_input，“ tags”：tags\_input }，ignore\_index = True）

tag\_another = 输入（ “标记另一个？键入（y）继续，或其他任何键退出。” ）

如果 tag\_another。lower（）== “ y”：

继续

中断

返回 df

new\_df = append\_to\_df（df）

What is the question?

Who is the manager?

Tags? Use commas to separate

customer\_support

Tag another? Type (y) to continue or any other key to exit.d

new\_df。头（n = 100）

<style scoped> .dataframe tbody tr th：only-of-type {vertical-align：middle; }

.dataframe tbody tr th {

vertical-align: top;

}

.dataframe thead th {

text-align: right;

}

</ style>

|  | **顾客** | **标签** |
| --- | --- | --- |
| **0** | 嗨，你明天几点钟开始... | [营业时间，营业时间] |
| **1个** | 我可以和你的经理谈谈吗？ | [客户支持] |
| **2** | 食物很棒，谢谢！ | [客户支持，反馈] |
| **3** | 您有什么类型的产品？ | [产品，菜单，库存，食品] |
| **4** | 您的厨房开放多久？ | [营业时间，营业时间，营业时间] |
| **5** | 我的订单准备不正确，我该如何... | [客户支持] |
| **6** | 你吃哪种肉？ | [菜单，产品，库存，食品] |
| **7** | 您的餐厅什么时候开放？ | [营业时间] |
| **8** | 您什么时候开晚餐？ | [营业时间，营业时间，营业时间] |
| **9** | 我该如何联系您？ | [联系人，customer\_support] |
| **10** | 谁是经理？ | [客户支持] |

new\_df。to\_pickle（“ dataset.pkl”）

**创建Rest API模型服务**

使用[fastapi](https://fastapi.tiangolo.com/)

！p ip install fastapi uvicorn 请求

Requirement already satisfied: fastapi in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (0.60.0)

Requirement already satisfied: uvicorn in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (0.11.6)

Requirement already satisfied: requests in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (2.24.0)

Requirement already satisfied: pydantic<2.0.0,>=0.32.2 in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (from fastapi) (1.6.1)

Requirement already satisfied: starlette==0.13.4 in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (from fastapi) (0.13.4)

Requirement already satisfied: websockets==8.\* in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (from uvicorn) (8.1)

Requirement already satisfied: uvloop>=0.14.0; sys\_platform != "win32" and sys\_platform != "cygwin" and platform\_python\_implementation != "PyPy" in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (from uvicorn) (0.14.0)

Requirement already satisfied: h11<0.10,>=0.8 in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (from uvicorn) (0.9.0)

Requirement already satisfied: httptools==0.1.\*; sys\_platform != "win32" and sys\_platform != "cygwin" and platform\_python\_implementation != "PyPy" in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (from uvicorn) (0.1.1)

Requirement already satisfied: click==7.\* in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (from uvicorn) (7.1.2)

Requirement already satisfied: chardet<4,>=3.0.2 in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (from requests) (3.0.4)

Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (from requests) (1.25.9)

Requirement already satisfied: idna<3,>=2.5 in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (from requests) (2.10)

Requirement already satisfied: certifi>=2017.4.17 in /Users/cfe/.local/share/virtualenvs/ml-hello-world-rt2goiiz/lib/python3.8/site-packages (from requests) (2020.6.20)

API\_APP\_PATH = 'app.py' ＃pathlib，os.path

＃从fastapi导入FastAPI

＃应用= FastAPI（）

＃@ app.get（“ /”）

＃def homepage\_view（）：

＃return {“ Hello”：“ World”}

％％WriteFile的 $ API\_APP\_PATH

进口 泡菜

从 fastapi 进口 FastAPI

从 pydantic 进口 BaseModel

应用 = FastAPI（）

model\_data = {}

与 开放（“model.pkl” ，'RB' ）作为 ˚F：

model\_data = 泡菜。负载（˚F。阅读（））

类 CustomerInput（BaseModel）：

查询：str

高清 预测（TXT = '世界，你好'，

向量化= 无，

型号= 无，

类= []，

\* ARGS，

\*\* kwargs）：

＃预解码

断言（矢量化！= 无）

断言（模型 ！= 无）

input\_vector = 矢量化。转换（[ txt]）

output\_vector = model。预测（input\_vector）

断言（len（output\_vector [ 0 ]）== len（classes））

preds = {}

for i，val in enumerate（output\_vector [ 0 ]）：

preds [ classes [ i ]] = int（val）

返回 Preds

@ app。后（“/预测”）

DEF predict\_view（customer\_input：CustomerInput）：

＃存储该查询数据- > SQL数据库

my\_pred = 预测（customer\_input。查询，\*\* model\_data）

返回 { “查询”：customer\_input。查询，“预测”：my\_pred }

＃@ app.post（'/ train'）

Overwriting app.py

汇入 要求

json = {

“ query”：“什么时候打开？

}

r = 请求。后（“http://127.0.0.1:8000/predict” ，JSON = JSON）

打印（ř。JSON（））

{'query': 'When do you open?', 'predictions': {'closing': 0, 'contact': 0, 'customer\_support': 0, 'feedback': 0, 'food': 0, 'hours': 1, 'inventory': 0, 'menu': 0, 'opening': 1, 'products': 0}}

＃label\_predictor\_from\_export（“您的厨房什么时候关闭？”，\*\* model\_loaded\_data）

**预测的回应**

bot\_responses = [

{

“响应”：[

“我们每天早上8点开放”，

“我们每天早上8点至晚上10点开放”，

“每天早上8点至晚上10点开放”

]，

“标签”：[ “小时”，“开放” ]

}，

{

“响应”：[

“炸玉米饼和汉堡”，

“比萨饼”

]，

“标签”：[ “菜单”，“食物” ]

}

]

bot\_df = pd。DataFrame（bot\_responses）

bot\_df。头（n = 100）

<style scoped> .dataframe tbody tr th：only-of-type {vertical-align：middle; }

.dataframe tbody tr th {

vertical-align: top;

}

.dataframe thead th {

text-align: right;

}

</ style>

|  | **回应** | **标签** |
| --- | --- | --- |
| **0** | [我们每天早上8点开放，我们早上8点开放... | [营业时间] |
| **1个** | [炸玉米饼和汉堡，比萨] | [菜单，食物] |

pred\_response = { 'query'：'什么时候打开？，'预测'：{ '关闭'：0，'联系'：0，'客户支持'：0，'反馈'：0，'食品'：0，'小时'：1，'库存'：0，'菜单'：0，'开幕'：1，'产品'：

pred\_tags = [ ķ 为 ķ，v 在 pred\_response [ '预测' ]。项目（）如果 v ！= 0 ]

pred\_tags

['hours', 'opening']

mask = bot\_df。标签。应用（lambda x：设置（pred\_tags）== 设置（x））

打印（遮罩）

0 True

1 False

Name: tags, dtype: bool

response\_df = bot\_df [ mask ] ＃bot\_df [bot\_df.tags.isin（“ abcs”）]

response\_df。头（）

<style scoped> .dataframe tbody tr th：only-of-type {vertical-align：middle; }

.dataframe tbody tr th {

vertical-align: top;

}

.dataframe thead th {

text-align: right;

}

</ style>

|  | **回应** | **标签** |
| --- | --- | --- |
| **0** | [我们每天早上8点开放，我们早上8点开放... | [营业时间] |

all\_responses = 列表（response\_df [ 'responses' ]。值）

print（all\_responses）

[['We open at 8am everyday', 'We are open from 8am to 10pm everyday', '8am to 10pm everyday']]

响应 = []

为 行 中 all\_responses：

对于 [R 在 排：

反应。追加（r）

响应 = 列表（设置（响应））

回应

['We are open from 8am to 10pm everyday',

'We open at 8am everyday',

'8am to 10pm everyday']

随机导入

高清 predict\_and\_respond（TXT = 无，bot\_df = 无）：

如果 TXT == 无 和 bot\_df 为 无：

回归 “对不起，我不知道是什么意思，请与我们联系。”

json = {

“ query”：txt

}

r = 请求。后（“http://127.0.0.1:8000/predict” ，JSON = JSON）

如果 ř。STATUS\_CODE 不是 在 范围（200，299）：

＃发送信号，记录

回报 。“对不起，我有现在的麻烦，请稍后再试。”

pred\_response = r。JSON（）

pred\_tags = [ ķ 为 ķ，v 在 pred\_response ['预测' ]。items（）如果 v ！= 0 ]

mask = bot\_df。标签。申请（拉姆达 X：组（pred\_tags）== 组（X））

response\_df = bot\_df [ 掩蔽 ]

all\_responses = 列表（response\_df [ '应答' ] 的值）

的响应 = []

为 行 在 all\_responses：

为 [R 在 行：

回应。append（r）

响应 = 列表（设置（响应）），

如果 len（响应）== 0：

返回 “对不起，我还在学习。我不明白你在说什么。”

返回 随机数。选择（回应）

Forecast\_and\_respond（“您明天上午9:30开放吗？”，bot\_df = bot\_df）

"Sorry, I am still learning. I don't understand what you said."

**下一步是什么？**

1. 获取更多数据，更多

* 立即添加更多数据。继续添加数据。
* 优化数据，移动标签，删除标签，升级。

1. 部署到生产服务器：

* 理想情况下，使用[此项目](https://www.codingforentrepreneurs.com/projects/serverless-container-python-app)使用FastAPI部署无服务器应用程序（就像我们所做的那样）。

1. 内部使用很多。

* 如果此工具成为在内部查找有关您的业务的答案的必备工具，那么它也可以成为外部（面向客户）的相同工具。如有疑问，请将其交给客户，

**关于**

通过此回购和教程系列，学习从头开始构建基本的机器学习模型。

[www.codingforentrepreneurs.com/projects/hello-world-machine-learning](https://www.codingforentrepreneurs.com/projects/hello-world-machine-learning)