机器学习新手使用入门

本文档介绍了如何使用机器学习按品种对鸢尾花进行分类（归类）。本文档深入介绍了如何使用 TensorFlow 代码执行此项任务，以此来解释机器学习的基本原理。

如果以下列表所描述的情况与您相符，那么您就来对地方了：

- 您对机器学习知之甚少或一无所知。
- 您想学习如何编写 TensorFlow 程序。
- 您可以使用 Python 进行编码（至少会一点）。

如果您已经熟悉机器学习的基本概念，但还不熟悉 TensorFlow，请阅读 TensorFlow 使用入门：面向机器学习专家 (https://www.tensorflow.org/get_started/premade_estimators)。

鸢尾花分类问题

想象一下，您是一名植物学家，正在寻找一种能够对所发现的每株鸢尾花进行自动分类的方法。机器学习提供多种花卉分类方法。例如，一个复杂的机器学习程序可以根据照片对花卉进行分类。我们的要求并不高 - 我们将仅根据鸢尾花花萼 (https://en.wikipedia.org/wiki/Sepal)和花瓣 (https://en.wikipedia.org/wiki/Petal)的长度和宽度对其进行分类。

鸢尾花约有 300 种，但我们的程序将仅对下列三种进行分类：

- 山鸢尾
- 维吉尼亚鸢尾
- 变色鸢尾



从左到右：*山鸢尾* (https://commons.wikimedia.org/w/index.php?curid=170298)**（提供者：Radomil** (https://commons.wikimedia.org/wiki/User:Radomil)**，依据 CC BY-SA 3.0 使用）、*变色鸢尾*** (https://commons.wikimedia.org/w/index.php?curid=248095)**（提供者：Dlanglois** (https://commons.wikimedia.org/wiki/User:Dlanglois)**，依据 CC BY-SA 3.0 使用）和*维吉尼亚鸢尾*** (https://www.flickr.com/photos/33397993@N05/3352169862)**（提供者：Frank Mayfield** (https://www.flickr.com/photos/33397993@N05)**，依据 CC BY-SA 2.0 使用）。**

幸运的是，有人已经创建了一个包含 120 株鸢尾花的数据集 (https://en.wikipedia.org/wiki/Iris_flower_data_set)（其中有花萼和花瓣的测量值）。该数据集已成为机器学习分类问题的标准入门内容之一。（MNIST 数据库 (https://en.wikipedia.org/wiki/MNIST_database)包含手写数字，是另一个热门的分类问题。）鸢尾花数据集的前 5 个条目如下所示：

花萼长度	花萼宽度	花瓣长度	花瓣宽度	品种
6.4	2.8	5.6	2.2	2
5.0	2.3	3.3	1.0	1
4.9	2.5	4.5	1.7	2
4.9	3.1	1.5	0.1	0
5.7	3.8	1.7	0.3	0

我们来介绍一些术语：

- 最后一列（品种）称为**标签** (https://developers.google.com/machine-learning/glossary/#label)；前四列称为**特征** (https://developers.google.com/machine-learning/glossary/#feature)。特征即某个样本的特点，而标签是我们尝试预测的内容。
- **样本** (https://developers.google.com/machine-learning/glossary/#example)由一个样本花卉的特征集和标签组成。上表显示了从包含 120 个样本的数据集中抽取的 5 个样本。

每个标签本质上是一个字符串（例如“setosa”），但机器学习通常依赖于数字值。因此，有人将每个字符串映射到一个数字。以下为表示方案：

- 0 代表 setosa（山鸢尾）
- 1 代表 versicolor（变色鸢尾）
- 2 代表 virginica（维吉尼亚鸢尾）

模型与训练

模型即特征与标签之间的关系。对于鸢尾花问题，模型定义了花萼和花瓣测量值与鸢尾花品种之间的关系。一些简单的模型可以用几行代数进行描述；比较复杂的机器学习模型则包含大量的交错数学函数和参数，以至于难以从数学角度进行总结。

您能否在不使用机器学习的情况下确定四个特征与鸢尾花品种之间的关系？也就是说，您能否使用传统编程技巧（例如大量条件语句）创建模型？或许可以。您可以反复研究数据集来确定花瓣和花萼测量值与特定品种的正确关系。不过，一个好的机器学习方法可为您确定模型。也就是说，如果您将足够多的代表性样本馈送到正确的机器学习模型类型中，该程序将确定花萼、花瓣与品种之间的关系。

训练是一种机器学习阶段，在此阶段中，模型会逐渐得到优化（不断学习）。鸢尾花问题是**监督式机器学习** (https://developers.google.com/machine-learning/glossary/#supervised_machine_learning)的一个示例，即模型通过包含标签的样本加以训练。（在**非监督式机器学习** (https://developers.google.com/machine-learning/glossary/#unsupervised_machine_learning)中，样本不包含标签。相反，模型通常会在特征中发现一些规律。）

获取示例程序

在使用本文档中的示例代码之前，请执行下列操作：

1. 安装 TensorFlow (https://www.tensorflow.org/install/index)。
2. 如果您是使用 virtualenv 或 Anaconda 安装的 TensorFlow，请激活您的 TensorFlow 环境。
3. 通过执行以下命令来安装或升级 Pandas：

   ```
   pip install pandas
   ```

执行下列步骤以获取示例程序：

1. 通过输入以下命令从 GitHub 克隆 TensorFlow 模型代码库：

   ```
   git clone https://github.com/tensorflow/models
   ```

2. 将此分支内的目录更改为包含本文档中所用示例的位置：

   ```
   cd models/samples/core/get_started/
   ```

在该 `get_started` 目录中，您会找到一个名为 `premade_estimator.py` 的程序。

运行示例程序

您可以像运行任何 Python 程序一样运行 TensorFlow 程序。因此，请从命令行执行以下命令来运行 **premade_estimators.py**：

```
python premade_estimator.py
```

运行程序后，系统应该会输出以三个预测行结尾的一大串信息，如下所示：

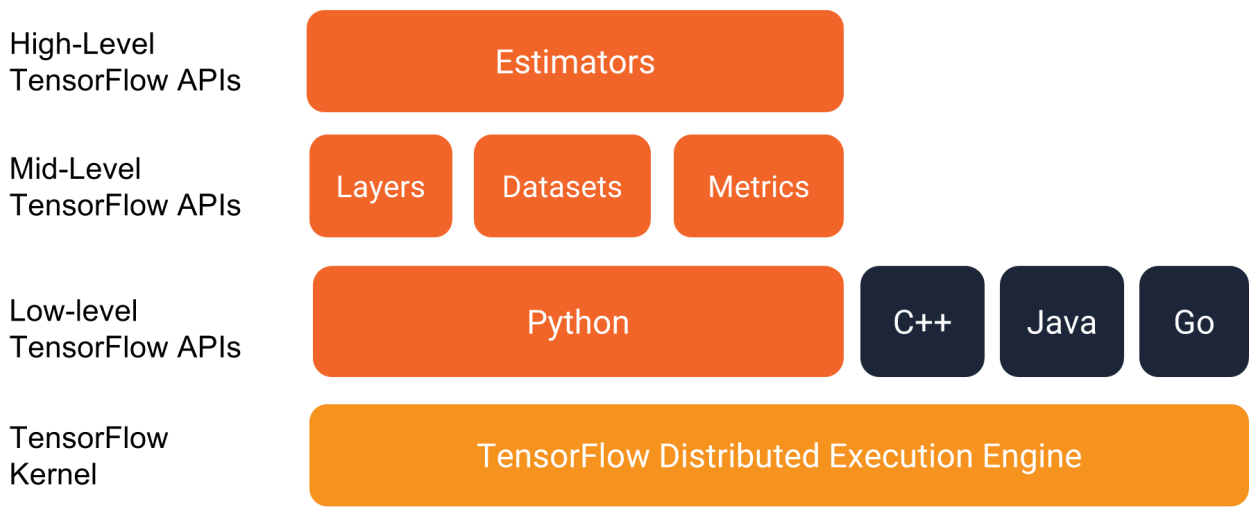
```
...
Prediction is "Setosa" (99.6%), expected "Setosa"

Prediction is "Versicolor" (99.8%), expected "Versicolor"

Prediction is "Virginica" (97.9%), expected "Virginica"
```

如果程序生成错误（而不是预测），请思考下列问题：

- 您是否正确安装了 TensorFlow?
- 您使用的 TensorFlow 版本是否正确？**premade_estimators.py** 程序需要 TensorFlow v1.4 或以上版本。
- 如果您是使用 virtualenv 或 Anaconda 安装的 TensorFlow，您是否激活了相应环境?

TensorFlow 编程堆栈

如下图所示，TensorFlow 提供一个包含多个 API 层的编程堆栈：



TensorFlow 编程环境。

在您开始编写 TensorFlow 程序时，我们强烈建议您重点了解下列两个高阶 API：

- Estimator
- Dataset

虽然我们偶尔会从其他 API 处获取便捷函数，但本文档重点介绍上述两个 API。

程序本身

感谢您的耐心等待；我们来深入了解代码。下面简要说明了 **premade_estimator.py** 以及很多其他 TensorFlow 程序：

- 导入和解析数据集。
- 创建特征列以描述数据。
- 选择模型类型。
- 训练模型。
- 评估模型的效果。
- 让经过训练的模型进行预测。

后续小节详细说明了每个部分。

导入和解析数据集

鸢尾花程序需要下列两个 .csv 文件中的数据：

- **http://download.tensorflow.org/data/iris_training.csv**：其中包含训练集。
- **http://download.tensorflow.org/data/iris_test.csv**：其中包含测试集。

训练集包含我们用于训练模型的样本；**测试集**包含我们用于评估训练后模型的效果的样本。

训练集和测试集起初是同一个数据集。然后，有人对样本进行拆分，大部分样本进入训练集，剩余部分进入测试集。向训练集添加样本通常会构建一个更好的模型；但是，向测试集添加更多样本则使我们能够更好地评估模型的效果。无论如何拆分，测试集中的样本都必须与训练集中的样本分隔开来。否则，您无法准确地确定模型的效果。

premade_estimators.py 程序依赖于 **load_data** 函数（位于相邻的 **iris_data.py** (https://github.com/tensorflow/models/blob/master/samples/core/get_started/iris_data.py) 文件中）来读取和解析训练集及测试集。下面是一个包含大量注释的该函数版本：

```
TRAIN_URL = "http://download.tensorflow.org/data/iris_training.csv"
TEST_URL = "http://download.tensorflow.org/data/iris_test.csv"

CSV_COLUMN_NAMES = ['SepalLength', 'SepalWidth',
                    'PetalLength', 'PetalWidth', 'Species']

...

def load_data(label_name='Species'):
    """Parses the csv file in TRAIN_URL and TEST_URL."""

    # Create a local copy of the training set.
    train_path = tf.keras.utils.get_file(fname=TRAIN_URL.split('/')[-1],
                                         origin=TRAIN_URL)
    # train_path now holds the pathname: ~/.keras/datasets/iris_training.csv

    # Parse the local CSV file.
    train = pd.read_csv(filepath_or_buffer=train_path,
                        names=CSV_COLUMN_NAMES,  # list of column names
                        header=0  # ignore the first row of the CSV file.
                       )
    # train now holds a pandas DataFrame, which is data structure
    # analogous to a table.

    # 1. Assign the DataFrame's labels (the right-most column) to train_label
    # 2. Delete (pop) the labels from the DataFrame.
    # 3. Assign the remainder of the DataFrame to train_features
    train_features, train_label = train, train.pop(label_name)

    # Apply the preceding logic to the test set.
    test_path = tf.keras.utils.get_file(TEST_URL.split('/')[-1], TEST_URL)
    test = pd.read_csv(test_path, names=CSV_COLUMN_NAMES, header=0)
    test_features, test_label = test, test.pop(label_name)

    # Return four DataFrames.
    return (train_features, train_label), (test_features, test_label)
```

Keras 是一个开放源代码机器学习库；**tf.keras** 是 Keras 的一种 TensorFlow 实现。**premade_estimator.py** 程序只访问一个 **tf.keras** 函数；即 **tf.keras.utils.get_file** 便捷函数，该函数会将远程 CSV 文件复制到本地文件系统。

调用 **load_data** 会返回两个 **(feature,label)** 对，分别对应训练集和测试集：

```
    # Call load_data() to parse the CSV file.
    (train_feature, train_label), (test_feature, test_label) = load_data()
```

Pandas 是一个开放源代码 Python 库，供多个 TensorFlow 函数使用。Pandas **DataFrame** (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html) 是一个包含已命名列标头和已编号行的表格。**load_data** 返回的特征都打包在 **DataFrames** 中。例如，**test_feature** DataFrame 如下所示：

```
    SepalLength  SepalWidth  PetalLength  PetalWidth
0           5.9         3.0          4.2         1.5
1           6.9         3.1          5.4         2.1
2           5.1         3.3          1.7         0.5
...
27          6.7         3.1          4.7         1.5
28          6.7         3.3          5.7         2.5
29          6.4         2.9          4.3         1.3
```

描述数据

特征列是一种数据结构，告知模型如何解读每个特征中的数据。在鸢尾花问题中，我们希望模型将每个特征中的数据解读为其字面浮点值；也就是说，我们希望模型将 5.4 这样的输入值解读为 5.4。不过，在其他机器学习问题中，通常需要以不太字面的方式解读数据。使用特征列解读数据是一个内容丰富的主题，我们将用整个文档 (https://www.tensorflow.org/get_started/feature_columns)来介绍它。

从代码角度看，您可以通过调用 **tf.feature_column** (https://www.tensorflow.org/api_docs/python/tf/feature_column) 模块中的函数来构建 **feature_column** 对象列表。每个对象都描述了模型的一个输入。要让模型将数据解读为浮点值，请调用 @{tf.feature_column.numeric_column)。在 **premade_estimator.py** 中，所有四个特征都应该解读为字面浮点值，因此用于创建特征列的代码如下所示：

```
# Create feature columns for all features.
my_feature_columns = []
for key in train_x.keys():
    my_feature_columns.append(tf.feature_column.numeric_column(key=key))
```

这是用于编写上述代码块的一个不太优雅，但可能更清晰的替代方法：

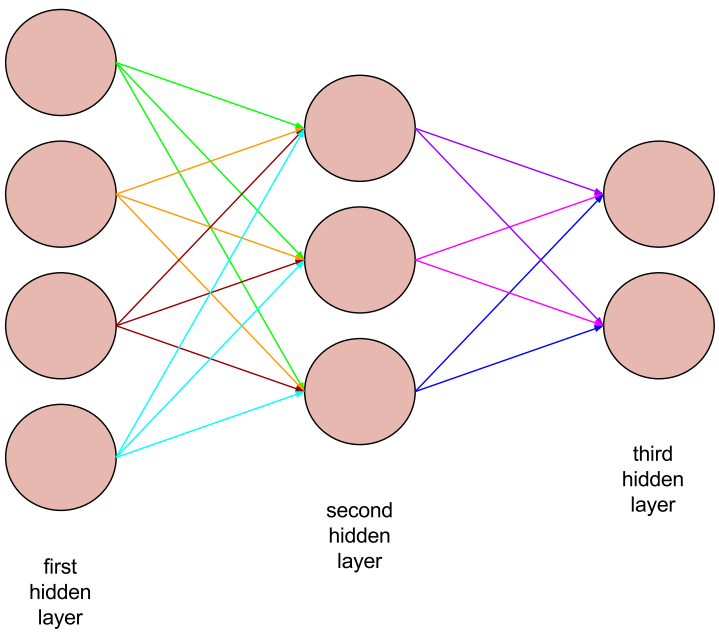
```
my_feature_columns = [
    tf.feature_column.numeric_column(key='SepalLength'),
    tf.feature_column.numeric_column(key='SepalWidth'),
    tf.feature_column.numeric_column(key='PetalLength'),
```

```
    tf.feature_column.numeric_column(key='PetalWidth')
]
```

选择模型类型

我们需要选择将要进行训练的模型类型。模型类型极其多，选择理想的类型需要经验。我们选择了一个神经网络来解决鸢尾花问题。**神经网络**(https://developers.google.com/machine-learning/glossary/#neural_network)可以发现特征与标签之间的复杂关系。神经网络是一个高度结构化的图，其中包含一个或多个**隐藏层**(https://developers.google.com/machine-learning/glossary/#hidden_layer)。每个隐藏层都包含一个或多个**神经元** (https://developers.google.com/machine-learning/glossary/#neuron)。神经网络存在多个类别。我们将使用**全连接神经网络**(https://developers.google.com/machine-learning/glossary/#fully_connected_layer)，这意味着一个层中的神经元将从上一层中的每个神经元获取输入。例如，下图显示了包含三个隐藏层的全连接神经网络：

- 第一个隐藏层包含四个神经元。
- 第二个隐藏层包含三个神经元。
- 第三个隐藏层包含两个神经元。



包含三个隐藏层的神经网络。

要指定模型类型，请实例化一个 **Estimator**(https://developers.google.com/machine-learning/glossary/#Estimators) 类。TensorFlow 提供了两

类 Estimator：

- **预创建的 Estimator**
  (https://developers.google.com/machine-learning/glossary/#pre-made_Estimator)：有人已为您编写完成。
- **自定义 Estimator**
  (https://developers.google.com/machine-learning/glossary/#custom_estimator)：必须自行编码（至少部分需要）。

为了实现神经网络，**premade_estimators.py** 程序会使用一个预创建的 Estimator（名为 **tf.estimator.DNNClassifier**
(https://www.tensorflow.org/api_docs/python/tf/estimator/DNNClassifier)）。此 Estimator 会构建一个对样本进行分类的神经网络。以下调用会实例化 **DNNClassifier**：

```
classifier = tf.estimator.DNNClassifier(
    feature_columns=my_feature_columns,
    hidden_units=[10, 10],
    n_classes=3)
```

使用 **hidden_units** 参数来定义神经网络内每个隐藏层中的神经元数量。为此参数分配一个列表。例如：

```
    hidden_units=[10, 10],
```

为 **hidden_units** 分配的列表长度表示隐藏层的数量（在此例中为 2 个）。列表中的每个值表示某个特定隐藏层中的神经元数量（第一个隐藏层中有 10 个，第二个隐藏层中有 10 个）。要更改隐藏层或神经元的数量，只需为 **hidden_units** 参数分配另一个列表即可。

隐藏层和神经元的理想数量取决于问题和数据集。与机器学习的多个方面一样，选择理想的神经网络形状需要一定的知识水平和实验基础。一般来说，增加隐藏层和神经元的数量通常会产生更强大的模型，而这需要更多数据才能有效地进行训练。

n_classes 参数指定了神经网络可以预测的潜在值的数量。由于鸢尾花问题将鸢尾花品种分为 3 类，因此我们将 **n_classes** 设置为 3。

tf.Estimator.DNNClassifier 的构造函数采用名为 **optimizer** 的可选参数，但我们的示例代码选择不指定该参数。**优化器**
(https://developers.google.com/machine-learning/glossary/#optimizer)会控制模型的训练方式。随着您在机器学习方面掌握的专业知识越来越丰富，优化器和**学习速率**
(https://developers.google.com/machine-learning/glossary/#learning_rate)将变得非常重要。

训练模型

实例化 **tf.Estimator.DNNClassifier** 会创建一个用于学习模型的框架。基本上，我们已经连接成一个网络，但还没有让数据从中流过。要训练神经网络，请调用 Estimator 对象的 **train** 方法。例如：

```
classifier.train(
    input_fn=lambda:train_input_fn(train_feature, train_label, args.batch
    steps=args.train_steps)
```

steps 参数指示 **train** 在完成指定的迭代次数后停止训练。增加 **steps** 会延长模型训练的时间。与直觉恰恰相反的是，训练模型的时间越长，并不能保证模型就越好。**args.train_steps** 的默认值是 1000。训练的步数是一个可以调整的**超参数** (https://developers.google.com/machine-learning/glossary/#hyperparameter)。选择正确的步数通常需要一定的经验和实验基础。

input_fn 参数会确定提供训练数据的函数。调用 **train** 方法表示 **train_input_fn** 函数将提供训练数据。下面是该方法的签名：

```
def train_input_fn(features, labels, batch_size):
```

我们将下列参数传递给 **train_input_fn**：

- **train_feature** 是 Python 字典，其中：
  - 每个键都是特征的名称。
  - 每个值都是包含训练集中每个样本的值的数组。
- **train_label** 是包含训练集中每个样本的标签值的数组。
- **args.batch_size** 是一个定义**批次大小** (https://developers.google.com/machine-learning/glossary/#batch_size)的整数。

train_input_fn 函数依赖于 **Dataset API**。这是一种高阶 TensorFlow API，用于读取数据并将其转换为 **train** 方法所需的格式。以下调用会将输入特征和标签转换为 **tf.data.Dataset** 对象，该对象是 Dataset API 的基类：

```
dataset = tf.data.Dataset.from_tensor_slices((dict(features), labels
```

tf.dataset 类提供很多用于准备训练样本的实用函数。以下行调用了三个此类函数：

```
dataset = dataset.shuffle(buffer_size=1000).repeat(count=None).batch
```

如果训练样本是随机排列的，则训练效果最好。要对样本进行随机化处理，请调用 **tf.data.Dataset.shuffle**。将 **buffer_size** 设置为大于样本数 (120) 的值可确保数据得

到充分的随机化处理。

在训练期间，**train** 方法通常会多次处理样本。在不使用任何参数的情况下调用 **tf.data.Dataset.repeat** 方法可确保 **train** 方法拥有无限量的训练集样本（现已得到随机化处理）。

train 方法一次处理一**批** (https://developers.google.com/machine-learning/glossary/#batch)样本。**tf.data.Dataset.batch** 方法通过组合多个样本来创建一个批次。该程序将默认**批次大小** (https://developers.google.com/machine-learning/glossary/#batch_size)设置为 100，意味着 **batch** 方法将组合多个包含 100 个样本的组。理想的批次大小取决于具体问题。一般来说，较小的批次大小通常会使 **train** 方法（有时）以牺牲准确率为代价来加快训练模型。

以下 **return** 语句会将一批样本传回调用方（**train** 方法）。

```
return dataset.make_one_shot_iterator().get_next()
```

评估模型

评估指的是确定模型进行预测的效果。要确定鸢尾花分类模型的效果，请将一些花萼和花瓣测量值传递给模型，并要求模型预测它们所代表的鸢尾花品种。然后，将模型的预测与实际标签进行比较。例如，如果模型对一半输入样本的品种预测正确，则准确率 (https://developers.google.com/machine-learning/glossary/#accuracy)为 0.5。下面展示了一个更有效的模型：

测试集

特征				**标签**	**预测**
5.9	3.0	4.3	1.5	1	1
6.9	3.1	5.4	2.1	2	2
5.1	3.3	1.7	0.5	0	0
6.0	3.4	4.5	1.6	1	2
5.5	2.5	4.0	1.3	1	1

一个准确率为 80% 的模型。

为评估模型的效果，每个 Estimator 都提供了 **evaluate** 方法。**premade_estimator.py** 程序会调用 **evaluate**，如下所示：

```
# Evaluate the model.
eval_result = classifier.evaluate(
    input_fn=lambda:eval_input_fn(test_x, test_y, args.batch_size))

print('\nTest set accuracy: {accuracy:0.3f}\n'.format(**eval_result))
```

调用 **classifier.evaluate** 与调用 **classifier.train** 类似。最大的区别在于，**classifier.evaluate** 必须从测试集（而非训练集）中获取样本。换言之，为了公正地评估模型的效果，用于评估模型的样本一定不能与用于训练模型的样本相同。**eval_input_fn** 函数负责提供来自测试集的一批样本。下面是 **eval_input_fn** 方法：

```
def eval_input_fn(features, labels=None, batch_size=None):
    """An input function for evaluation or prediction"""
    if labels is None:
        # No labels, use only features.
        inputs = features
    else:
        inputs = (features, labels)

    # Convert inputs to a tf.dataset object.
    dataset = tf.data.Dataset.from_tensor_slices(inputs)

    # Batch the examples
    assert batch_size is not None, "batch_size must not be None"
    dataset = dataset.batch(batch_size)

    # Return the read end of the pipeline.
    return dataset.make_one_shot_iterator().get_next()
```

简而言之，当由 **classifier.evaluate** 调用时，**eval_input_fn** 会执行以下操作：

1. 将测试集中的特征和标签转换为 **tf.dataset** 对象。
2. 创建一批测试集样本。（无需随机化处理或重复使用测试集样本。）
3. 将该批次的测试集样本返回 **classifier.evaluate**。

运行此代码会生成以下输出（或类似输出）：

```
Test set accuracy: 0.967
```

0.967 的准确率表示我们经过训练的模型正确分类了测试集中 30 个鸢尾花品种中的 29 个品种。

预测

我们现已训练了一个模型并“证明”它是有效的，但在对鸢尾花品种进行分类方面，这还不够。现在，我们使用经过训练的模型对**无标签样本** (https://developers.google.com/machine-learning/glossary/#unlabeled_example)（即包含特征但不包含标签的样本）进行一些预测。

在现实生活中，无标签样本可能来自很多不同的来源，其中包括应用、CSV 文件和数据 Feed。现在，我们直接手动提供下列三个无标签样本：

```
    predict_x = {
        'SepalLength': [5.1, 5.9, 6.9],
        'SepalWidth': [3.3, 3.0, 3.1],
        'PetalLength': [1.7, 4.2, 5.4],
        'PetalWidth': [0.5, 1.5, 2.1],
    }
```

每个 Estimator 均提供一个 **predict** 方法，**premade_estimator.py** 通过如下方式调用该方法：

```
predictions = classifier.predict(
    input_fn=lambda:eval_input_fn(predict_x, batch_size=args.batch_size))
```

与 **evaluate** 方法一样，我们的 **predict** 方法也收集来自 **eval_input_fn** 方法的样本。

在进行预测时，我们没有将标签传递给 **eval_input_fn**。因此，**eval_input_fn** 将执行以下操作：

1. 转换我们刚刚手动创建的 3 元素集合中的特征。
2. 根据该手动集合创建一个包含 3 个样本的批次。
3. 将该批样本返回 **classifier.predict**。

predict 方法返回一个 Python 可迭代对象，为每个样本生成一个预测结果字典。此字典包含几个键。**probabilities** 键存储的是一个由三个浮点值组成的列表，每个浮点值表示输入样本是特定鸢尾花品种的概率。以下面的 **probabilities** 列表为例：

```
'probabilities': array([  1.19127117e-08,   3.97069454e-02,   9.6029299
```

上述列表表明：

- 该鸢尾花为山鸢尾的可能性微乎其微。
- 该鸢尾花为变色鸢尾的几率为 3.97%。
- 该鸢尾花为维吉尼亚鸢尾的几率为 96.0%。

class_ids 键存储的是一个 1 元素数组，用于标识可能性最大的品种。例如：

```
'class_ids': array([2])
```

数字 **2** 对应维吉尼亚鸢尾。以下代码会遍历返回的 **predictions**，以报告每个预测：

```
for pred_dict, expec in zip(predictions, expected):
    template = ('\nPrediction is "{}" ({:.1f}%), expected "{}"')

    class_id = pred_dict['class_ids'][0]
    probability = pred_dict['probabilities'][class_id]
    print(template.format(SPECIES[class_id], 100 * probability, expec))
```

运行该程序会产生以下输出：

```
...
Prediction is "Setosa" (99.6%), expected "Setosa"

Prediction is "Versicolor" (99.8%), expected "Versicolor"

Prediction is "Virginica" (97.9%), expected "Virginica"
```

总结

本文档简要介绍了机器学习。

由于 **premade_estimators.py** 依赖于高阶 API，因此机器学习中的大部分复杂数学概念都暂且忽略不谈。如果您打算熟练掌握机器学习，我们建议您详细了解**梯度下降法** (https://developers.google.com/machine-learning/glossary/#gradient_descent)、批处理和神经网络。

我们建议您接下来阅读特征列 (https://www.tensorflow.org/get_started/feature_columns)文档，它介绍了如何在机器学习中表示不同类型的数据。

上次更新日期：四月 19, 2018