1、介绍鸢尾花分类问题
适用于：
- 对机器学习几乎一无所知
- 想要学习TensorFlow编程
- 至少会一点Python

熟悉机器学习基本概念的TensorFlow新人可以跳去阅读：Getting Started with TensorFlow: for ML Experts。

2、鸢尾花分类问题
- 根据萼片和花瓣的长度和宽度来对鸢尾花进行分类
- 鸢尾花三种类别：（1）Iris setosa（2）Iris virginica（3）Iris versicolor
- 鸢尾花数据集：120条数据

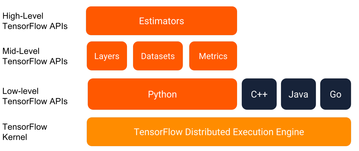
Sepal length	sepal width	petal length	petal width	species
6.4	2.8	5.6	2.2	2
5.0	2.3	3.3	1.0	1
4.9	2.5	4.5	1.7	2
4.9	3.1	1.5	0.1	0
5.7	3.8	1.7	0.3	0

- 类别标签编码：0—setosa, 1—versicolor，2—virginica
- 四列特征（花萼长度，花萼宽度，花瓣长度，花瓣宽度），一列标签

3、模型与训练
- 监督学习，非监督学习

4、获取示例程序
- 安装tensorflow
- 安装pandas
- 示例程序：https://github.com/tensorflow/models
- 目录：models/samples/core/get_started/premade_estimator.py
- 运行 premade_estimator.py 程序

5、TensorFlow编程堆栈
- TensorFlow编程环境



- 重点关注的高级APIs：
  - Estimators：代表一个完整的模型，提供训练、评估、预测的方法
  - Datasets：构建数据输入管道，提供加载和操作数据的方法

6、程序本身
- TensorFlow程序的一般轮廓
  - 导入和解析数据集
  - 创建feature columns描述数据
  - 选择模型的类型
  - 训练模型
  - 评估模型有效性
  - 预测

7、导入和解析数据集
- 训练集：iris_training.csv
- 测试集：iris_test.csv
- 下载数据：tf.keras
- 利用Pandas打包数据

```
     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
```

8、描述数据（Describe the data）
- feature column是一种数据结构，用来告诉模型如何解释数据的每个特种。
- 例如，在鸢尾花分类问题中，我们希望模型将每个特征中的数据解释为其字面上的浮点值
- Feature Columns
- tf.feature_column：构建feature_column对象列表，每个对象描述一个输入
- tf.feature_column.numeric_column：告诉模型将数据解释为浮点值
- 创建feature column

```
# 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))
```

- 等价于：

```
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")
]
```

8、选择模型的类型
- 全连接神经网络
- 要指定一个模型类型，需要实例化一个Estimator类
- TensorFlow提供了两类Estimators：
  - pre-made Estimators：预创建好的
  - custom Estimators：需要自定义的
- 使用预创建的：tf.estimator.DNNClassifier

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

- hidden_units=[10, 10]，定义每个隐层的单元数
- n_classes=3，类别数
- optimizer，可选参数（优化器控制模型如何训练），默认是'Adagrad'
- 学习率

9、训练模型
- 实例化之后，调用train方法

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

- steps=args.train_steps：训练迭代次数，超参数
- input_fn：提供训练数据的方法

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

- train_feature：python字典，每个key是一个特征名，每个value是数据值的数组
- train_label：标签值的数组
- args.batch_size：batch大小，整数
- train_input_fn方法依赖于Dataset API（一个高级TensorFlow API，用于读取数据和转换数据格式）
- 下面的调用将输入特征和标签转换成一个tf.data.Dataset对象，它是Dataset API的一个基类

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

- tf.dataset类：提供许多预处理训练数据的方法

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

- tf.data.Dataset.shuffle：将训练数据的顺序随机化
- 设置buffer_size参数 > 数据个数
- tf.data.Dataset.repeat：重复生成数据，不指定参数的话则为无限次
- tf.data.Dataset.batch：连接多个数据生成batch，通常较小的batch size训练更快（有时牺牲准确率）
- 返回一个batch的数据：

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

10、评估模型
- 调用评估方法：在测试集上

```
# 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))
```

- eval_input_fn方法：提供一个batch的测试数据
- 测试集数据不需要随机化顺序和重复生成

```
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()
```

- Test set accuracy: 0.967

11、预测
- 3条新数据

```
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],
}
```

- 调用预测方法：

```
predictions = classifier.predict(
    input_fn=lambda:eval_input_fn(predict_x,
                                  labels=None,
                                  batch_size=args.batch_size))
```

- eval_input_fn方法：注意新数据没有标签
- 提供一个batch的新数据供预测
- 预测方法返回一个迭代器结果，每条数据结果的内容有：

```
'probabilities': array([  1.19127117e-08,   3.97069454e-02,   9.60292995e-01])

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

- 输出每条预测：

```
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(iris_data.SPECIES[class_id], 100 * probability, expec))
```

12、总结
- 本文提供机器学习简介
- 示例的高级API隐藏了机器学习的许多数学复杂性
- 因此推荐学习更多关于梯度下降、批处理（batching）和神经网络的知识
- 推荐阅读Feature Columns文档，关于如何表示机器学习中不同类型的数据