本文档介绍TensorFlow编程环境。演示如何用TensorFlow解决鸢尾花分类问题。

- 要求：安装tensorflow和pandas

1、获取示例程序

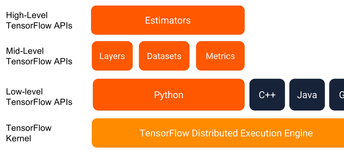
- 示例程序：https://github.com/tensorflow/models
- 目录：models/samples/core/get_started/premade_estimator.py
- 运行 premade_estimator.py 程序
- 输出：

```
...

Prediction is "Setosa" (99.6%), expected "Setosa"
Prediction is "Versicolor" (99.8%), expected "Versicolor"
Prediction is "Virginica" (97.9%), expected "Virginica"
```

2、TensorFlow编程堆栈

- TensorFlow编程环境



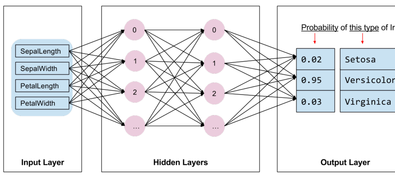
- 重点关注的高级APIs：
  - Estimators：代表一个完整的模型，提供训练、评估，预测的方法
  - Datasets：构建数据输入管道，提供加载和操作数据的方法

3、鸢尾花分类：概述

- 根据萼片和花瓣的长度和宽度来对鸢尾花进行分类
- 四列特征（花萼长度、花萼宽度，花瓣长度、花瓣宽度），一列标签
- 类别标签编码：0—setosa, 1—versicolor, 2—virginica
- 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

- 算法：
  - Deep Neural Network classifier
  - 两个隐层，每个隐层有10个隐藏单元节点



- 推理（Inference）：得到各个类别的概率

4、Estimators编程概述

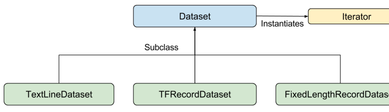
- Estimator：一个完整模型的表示，处理初始化，日志记录，保存和恢复等细节
- 一个Estimator是源于tf.estimator.Estimator的类
- TensorFlow提供了一组预先写好的Estimators（例如LinearRegressor）来实现机器学习算法
- 也可以编写自己的自定义Estimators
- 基于pre-made Estimators的TF编程需要完成以下任务：
  - 创建一个或多个输入函数
  - 定义模型的feature columns
  - 初始化一个Estimator，指定feature columns和各种超参数
  - 在Estimator对象上调用一个或多个方法，传入适当的输入函数作为数据源

5、创建输入函数

- 用于提供数据
- 输入函数返回一个tf.data.Dataset对象，输出下列二元组：
  - 特征：python字典，每个key是一个特征名，每个value是所有特征值的数组
  - 标签：包含所有标签值的数组
- 输入函数示例

```
def input_evaluation_set():
    features = {'SepalLength': np.array([6.4, 5.0]),
                'SepalWidth':  np.array([2.8, 2.3]),
                'PetalLength': np.array([5.6, 3.3]),
                'PetalWidth':  np.array([2.2, 1.0])}
    labels = np.array([2, 1])
    return features, labels
```

- 输入函数可以有多种方式，推荐用Dataset API
- Dataset API包含下列类：
  - Dataset：基类，包含创建和转换数据集的方法，可以从内存或python生成器中初始化数据集
  - TextLineDataset：从文本文件中读取行
  - TFRecordDataset：从TFRecord文件中读取记录
  - FixedLengthRecordDataset：从二进制文件中读取固定大小的记录
  - Iterator：提供了一次访问一个数据集元素的方法



- 使用Dataset API能并行地读取大量文件集合，并连接到一个数据流中
- 输入函数（位于iris_data.py）：

```
def train_input_fn(features, labels, batch_size):
    """An input function for training"""
    # Convert the inputs to a Dataset.
    dataset = tf.data.Dataset.from_tensor_slices((dict(features), labels))

    # Shuffle, repeat, and batch the examples.
    return dataset.shuffle(1000).repeat().batch(batch_size)
```

6、定义特征列（feature columns）

- 特征列是一个对象，描述了模型如何使用特征字典里的原始输入数据
- 当构建一个Estimator模型时，需要给模型传递一个特征列列表，告诉模型如何使用每个特征
- tf.feature_column模块提供了许多表示数据的选项
- 例如，将鸢尾花数据集4个原始特征表示为32位浮点型数值：

```
# Feature columns describe how to use the input.
my_feature_columns = []
for key in train_x.keys():
    my_feature_columns.append(tf.feature_column.numeric_column(key=key))
```

7、实例化一个estimator

- 预置分类器Estimators：
  - tf.estimator.DNNClassifier：多类分类的深度模型
  - tf.estimator.DNNLinearCombinedClassifier：for wide & deep 模型
  - tf.estimator.LinearClassifier：基于线性模型的分类器
- 实例化一个DNNClassifier Estimator

```
# Build a DNN with 2 hidden layers and 10 nodes in each hidden layer.

classifier = tf.estimator.DNNClassifier(
    feature_columns=my_feature_columns,
    # Two hidden layers of 10 nodes each.
    hidden_units=[10, 10],
    # The model must choose between 3 classes.
    n_classes=3)
```

8、训练，评估，预测

- 有了Estimator对象之后，就可以调用方法了：
  - 训练模型
  - 评估模型
  - 执行预测
- 训练模型：调用train方法

```
# Train the Model.
classifier.train(
    input_fn=lambda:iris_data.train_input_fn(train_x, train_y, args.batch_size),
    steps=args.train_steps)
```

- 使用lambda
- steps：训练步数
- 评估模型（在测试集上）：

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

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

- eval_input_fn：只生成一个epoch的数据
- Test set accuracy: 0.967
- 执行预测：

```
# Generate predictions from the model
expected = ['Setosa', 'Versicolor', 'Virginica']
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],
}

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

- 输出预测结果：

```
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))
```

9、总结

- 使用Pre-made Estimators可以快速创建标准模型
- 继续阅读：
  - Checkpoints（检查点）：保存和恢复模型
  - Datasets（数据集）：将数据导入模型
  - Creating Custom Estimators（创建自定义Estimators）：针对特定问题自定义Estimator

更新时间：2018年04月20日10:06:03