Step by Step，一个周末入门机器学习

v1.0.0 (2017-12-28)

    - 开始编码
    - 运行结果
    -
- 总结
    - 要点回顾
    - 错过了什么
    - 那接下来呢

导读

几个月前开始入门机器学习时，面对一堆英文视频和数学基础，咬紧牙关摸爬滚打，终于可以在SideProject上试着使用机器学习了。

回首前路，发现自己摸索的是一条坑多路绕的林间小道，而当我到达此处，发现并不需要如此艰苦耗时，作为前车之鉴，将这几个月的心得分享给刚开始入门或正打算入门，以及从入门到放弃了的广大程序员。

本系列文章面向对机器学习感兴趣的任何人，无需数学基础，力求从无任何基础的角度与实践方式出发，让大家能够在几天的时间内了解最简的机器学习使用流程，以实验的性质运用到工作或个人项目中，解决一些简单但繁琐的实际问题，从而为更进一步的深入学习打下基础。

本文会做不定期修改，请到原文链接下载最新版。

基础理论

关于机器学习

回看历史，当科技发展到一定程度的时候，很可能会对社会的发展变革产生显著的影响。

- 第一次科技革命，随着蒸汽机、采煤、钢铁等技术的发展，人类大规模的用机器取代了人力、兽力的生产；
- 第二次科技革命，随着电力得到了大规模的应用，极大地推动了生产力的发展；
- 第三次科技革命，随着信息技术的发展，人类进入了数字化的信息时代。

当下，学术研究和硬件性能所达到的高度，很可能使得机器学习成为第四次科技革命的开始。

曾经技术娴熟的手工艺人，在自动化设备的质量和效率面前只能兴叹；而现在处于世界围棋巅峰的人类，AlphaGo只要自学3天就能将其打败。

所以说机器学习的本质其实和蒸汽机，电力一样，它是一种技术，一种可以扩展人类自身限制的技术。

例如开车，你并不需要了解汽车的所有原理，只要学会了驾驶技术，有了驾照，就能使用汽车来打破自身速度和体能的限制，在短时间内到达以前步行所不能及的远方；机器学习也是一样的，你并不需要学会所有相关的理论知识，只要掌握了一定的概念和流程，就能用它来打破自己大脑的限制，产生出以前自身智力无法企及的解决问题的方法。

当然，会修车的驾驶员能在汽车出现问题的时候找到问题并解决，这些会汽车修理的驾驶员后来就成为了汽车方面的专家；想要成为机器学习的专家，不仅要学会如何使用，还要深刻理解其运作的原理，并且能够调整和改进它。

本文致力于了解如何使用机器学习而不考虑其背后的原理，在不使用任何图像识别技术的前提下，通过一个手写数字识别App的开发，来感受机器学习给我们带来的更多可能。

什么是机器学习

抽象的说法，通多对旧数据的分析，自动总结出能预测在新条件下结果的方法。

思考

机器学习和人类学习的区别和各自的优劣势。

机器学习流程

先来想一下我们人类的学习过程是怎样的?

一个自然风光的摄影师，常年在大自然中拍摄取景，经过多年的观察与积累，成为了一个预测天气的好手。

而计算机是怎样成为天气预测专家的呢？输入了大量的历史数据，经过分析对比，找到了规律，天气预测的结果八九不离十，准确率比摄影师还要高。

思考

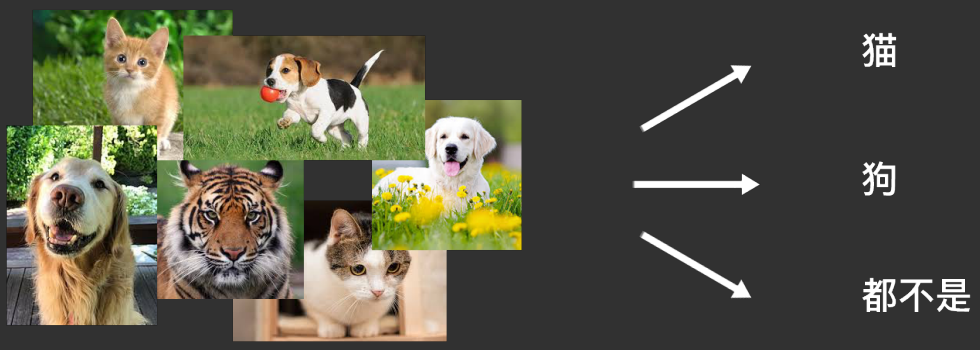
他们的相同点是都需要经验累积，而差异就在于人类需要数年甚至数十年才能成为某个领域的专家，而机器只要有足够的数据，只要很短的时候就能达到或超越人类。

机器学习分类

目前我们只要关心两种机器学习的类型：**预测** 和 **分类**。

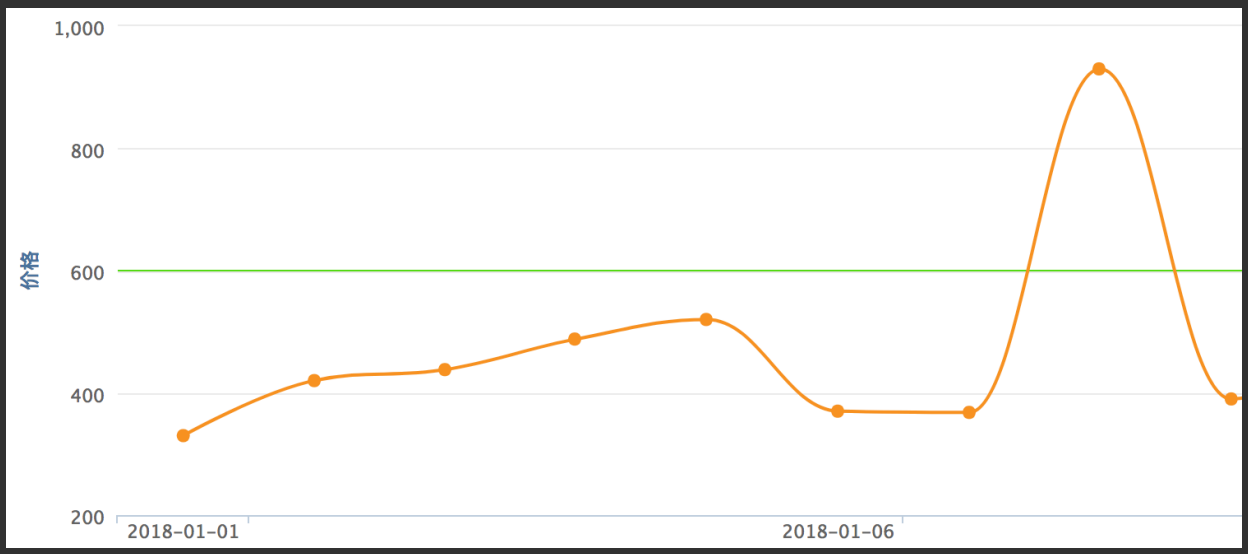
分类

分类的结果是离散的，或者说是枚举类型，例如一张图片，识别其中是 1猫 2狗 3两者都不是，结果里有3个枚举，所以属于分类



预测

相对于分类，预测（或者叫线性回归）的结果是连续的，在一个区间范围内的某一点，例如给定日期和航班，预测机票的价格，它的结果是一个在0到正无穷（从理论上来说）的开区间上一点。



总结

回顾一下机器学习的过程

- 首先，需要收集足够的数据；
- 然后，计算机需要用某种方法对数据进行分析学习；
- 最后，机器具备了预测或分类的能力。

以下问题属于分类还是预测?

- 通过用户的历史浏览及购买记录，判断他是否会购买某个商品。
- 为用户上传的图片自动打上标签。
- 预报明天的PM2.5指数

下一章将开始机器学习的实践，需要具备初级的编程能力。

Hello World

环境搭建

Anaconda

Anaconda是一个非常方便的数据分析Python集成工具/环境，包含了大部分数据分析所用的工具/包，支持Mac，Windows，Linux。我们只需安装这一个软件即可。

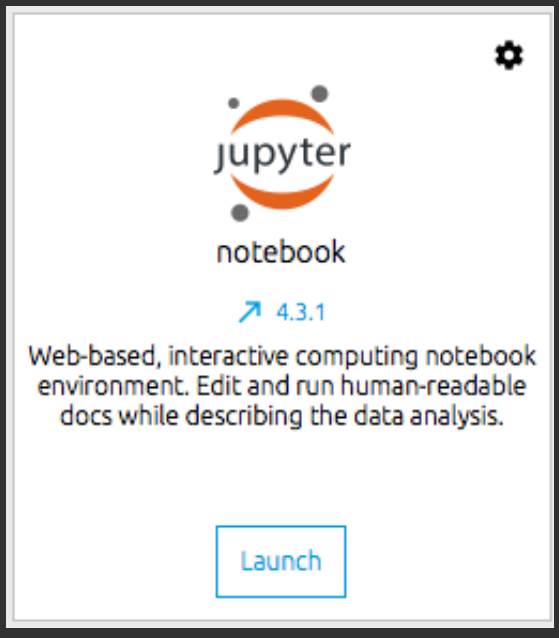
如果你已经熟悉Python的开发可略过直接看下一节。

下载Anaconda（不清楚版本区别的话建议安装Python3.6的版本）

https://www.anaconda.com/download

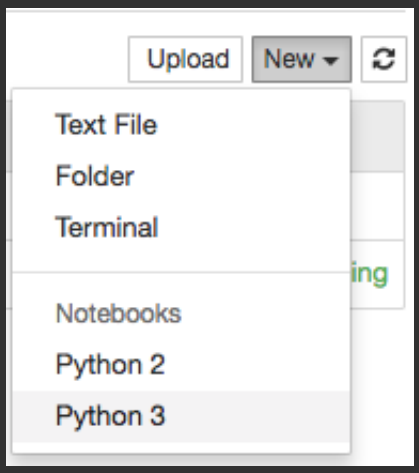
Mac安装使用默认配置即可，Windows需要手动选中配置环境变量。

完成后运行 Anaconda Navigator。



运行 Jupyter notebook，会自动打开一个本地服务器的网页，进入AnacondaProjects文件夹。

新建一个Python3的文件：



新打开的页面是一个交互式的Python编程环境，可以在其中运行代码并实时输出结果，非常方便。

一切就绪，可以开始编码了，以下操作均为Mac环境，其他环境大同小异，若有差别请自行Google。

学习目标

在本节中，我们首先要训练一个可识别手写数字的模型，然后把模型保存下来。

这是一套最简化的将机器学习使用到实际应用中的流程的第一步，也是将机器学习应用到实践中最重要的一步，能让我们对机器学习中最重要的概念之一：**训练**，有一个基本的认识。

数据准备

可在交互式环境中逐块输入以下代码，Shift + 回车 运行。

上一章中说到机器学习的第一步是收集数据，我们就使用这套已经收集好的数据集。

导入函数库

```
import matplotlib.pyplot as plt
from sklearn import datasets, svm
```

- **matplotlib.pyplot** : 用来显示图像
- **sklearn.datasets** : 包含了各种类型大量的机器学习数据集
- **sklearn.svm** : SVM(Support Vector Machine)支持向量机，机器学习中的一种算法
- **sklearn.metrics** : 用来评估模型的准确率

加载数据

```
digits = datasets.load_digits()
```

- **load_digits()** : sklearn中包含的一组手写数字的数据集，可看作是机器学习中的Hello World

数据信息

- **images** : 8x8的矩阵二维数组，存储了手写数字的图像数值
- **data** : 将images中的矩阵摊平成64个元素的一维数组
- **target** : images中的每个矩阵所代表的数字，0-9
- **target_names** : target的标签

可以通过逐条输出来自己熟悉一下整个数据的结构

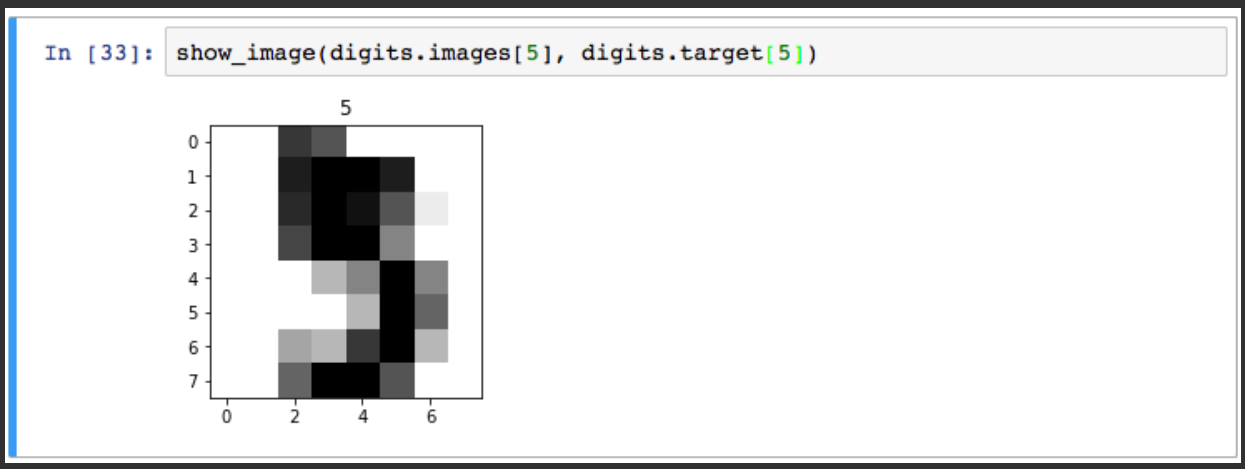
```
len(digits.images)
digits.images[5]
digits.data[5]
digits.target[5]
digits.target_names[5]
```

接下来我们定义一个函数，可以图形化的显示数据

```
def show_image(image, label):
    plt.figure(1, figsize=(3, 3))
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.title(label)
    plt.show()
```

现在我们可以使用这个函数来更加直观的显示数据，随便显示几个试试

```
show_image(digits.images[5], digits.target[5])
```



训练模型

数据已经准备好，我们可以开始训练模型了

```
# 对数据进行分割，使用2/3的数据进行训练，1/3的数据进行测试
# train_test_split是sklearn提供的一个用于分割数据的函数
# X_train : 训练用的图片数据
# y_train : 训练用的已分类的图片类别
# X_train : 测试用的图片数据
# y_train : 测试时用于对比测试结果的已分类图片类别
X = digits.data
y = digits_dataset.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

# 初始化一个SVC (Support Vector Classification) 分类器
classifier = svm.SVC(gamma=0.001)

# 使用训练数据进行训练
classifier.fit(X_train, y_train)

# 查看模型在测试数据上的预测准确度
classifier.score(X_test, y_test)
# 0.963272120200333393

# 对比下前20个中的预测结果
predicted = classifier.predict(y_test[:20])
print(predicted)
print(testing_target[:20])

# 20个中有一个错误,准确率95%，还是基本符合上面0.963的准确率
```

```
In [38]: print(predicted)
         print(testing_target[:20])

         [4 1 7 7 8 5 1 0 0 2 2 7 8 2 0 1 2 6 3 3]
         [4 1 7 7 3 5 1 0 0 2 2 7 8 2 0 1 2 6 3 3]
```

- 训练数据和测试数据不能重合，就好像平时训练我们做了很多真题，而考试的时候出现的都是已经做过的题目，这样就不能很好的反映出我们真实的水平

保存模型

上面我们所使用的训练数据是一个很小的数据集，很快就把模型训练完了，但在真实的环境中，一般情况下的训练数据是很庞大的，并不能很快训练完成，有的需要几十分钟，几个小时，甚至几天。所以我们需要把训练好的模型保存下来，以备下次使用。

sklearn已经包含了保存模型的模块joblib

```
from sklearn.externals import joblib
```

保存

```
joblib.dump(classifier, 'digits.pkl')
```

读取

```
classifier = joblib.load('digits.pkl')
```

要点自查

- 如何使用sklearn中的数据集
- 如何使用数据训练模型
- 如何保存已训练好的模型

First App

学习目标

本节会将上一节中导出的模型使用到我们的iOS工程中，开发一个可通过摄像头拍照进行手写数字识别的App。

本节开发的例子是基于iOS的，如果你是其他平台的开发者，可查阅对应的资料，将模型应用于自己熟悉的平台上。

模型转换

从iOS11开始，Apple推出的CoreML库支持了机器学习，能很方便的将训练好的模型集成到Xcode工程中，但前提是需要将训练好的模型转换成Xcodo支持的格式（.mlmodel）。

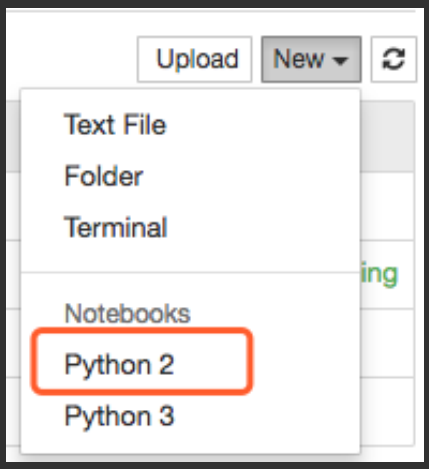
使用Python2

模型转换使用的是Apple提供的一个基于Python的工具coremltools，但目前仅支持Python2，如果使用的是3的话，还需要安装一个支持2的模块，命令行中直接运行：

```
python2 -m pip install ipykernel
python2 -m ipykernel install --user
```

参考链接 https://stackoverflow.com/questions/30492623/using-both-python-2-x-and-python-3-x-in-ipython-notebook

安装完成后，新建一个Python2的文件：



由于我们上一节使用的是Python3导出的模型，在Python2是不能直接读取的，所以我们需要将上一节的内容在Python2上再做一遍（不要直接复制运行，自己写一遍）：

```
from sklearn import datasets, svm
from sklearn.model_selection import train_test_split

digits = datasets.load_digits()

# 对数据进行分割，使用2/3的数据进行训练，1/3的数据进行测试
# train_test_split是
# X_train : 训练用的图片数据
# y_train : 训练用的已分类的图片类别
# X_train : 测试用的图片数据
# y_train : 测试时用于对比测试结果的已分类图片类别
X = digits.data
y = digits.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

# 初始化一个SVC (Support Vector Classification) 支持向量机分类器
classifier = svm.SVC(gamma=0.001)

# 使用训练数据进行训练
classifier.fit(X_train, y_train)

# 查看模型在测试数据上的预测准确度
score = classifier.score(X_test, y_test)
print(score)
# 0.96327212020033393

# 保存模型
joblib.dump(classifier, 'digits.pkl')
```

你可能会问，为什么不一开始就使用Python2呢?

- 重写一遍代码进行复习
- 自查资料进一步了解Python2和3的区别
- Python3是未来，也许你看到的时候coremltools已经支持了呢

转换

先在命令行中安装一下coremltools

```
pip install -U coremltools
```

现在我们可以将保存的模型转换为Xcode支持的格式了：

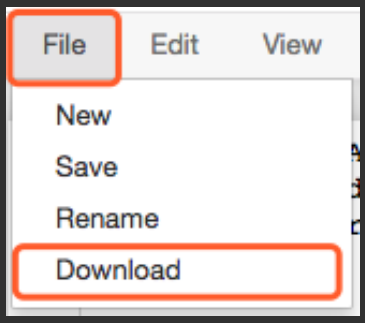
```
from sklearn.externals import joblib
import coremltools

# 读取训练好的模型
classifier = joblib.load('digits.pkl')

# 生成模型的输入参数名称，总共有64个参数，所以我们生成的参数名称如下
# feature_0, feature_1 ... feature_63
feature_names = ["feature_"+str(i) for i, x in enumerate(X_train[0])]

# 将模型转换为coreml的格式，并保存下来
coreml_model = coremltools.converters.sklearn.convert(classifier,
feature_names, "digit")
coreml_model.save("digits.mlmodel")
```

现在可以在列表中看到digits.mlmodel这个文件了，点击后会在新的页面打开，由于是特殊的格式无法预览，选择 File -> Download 下载下来

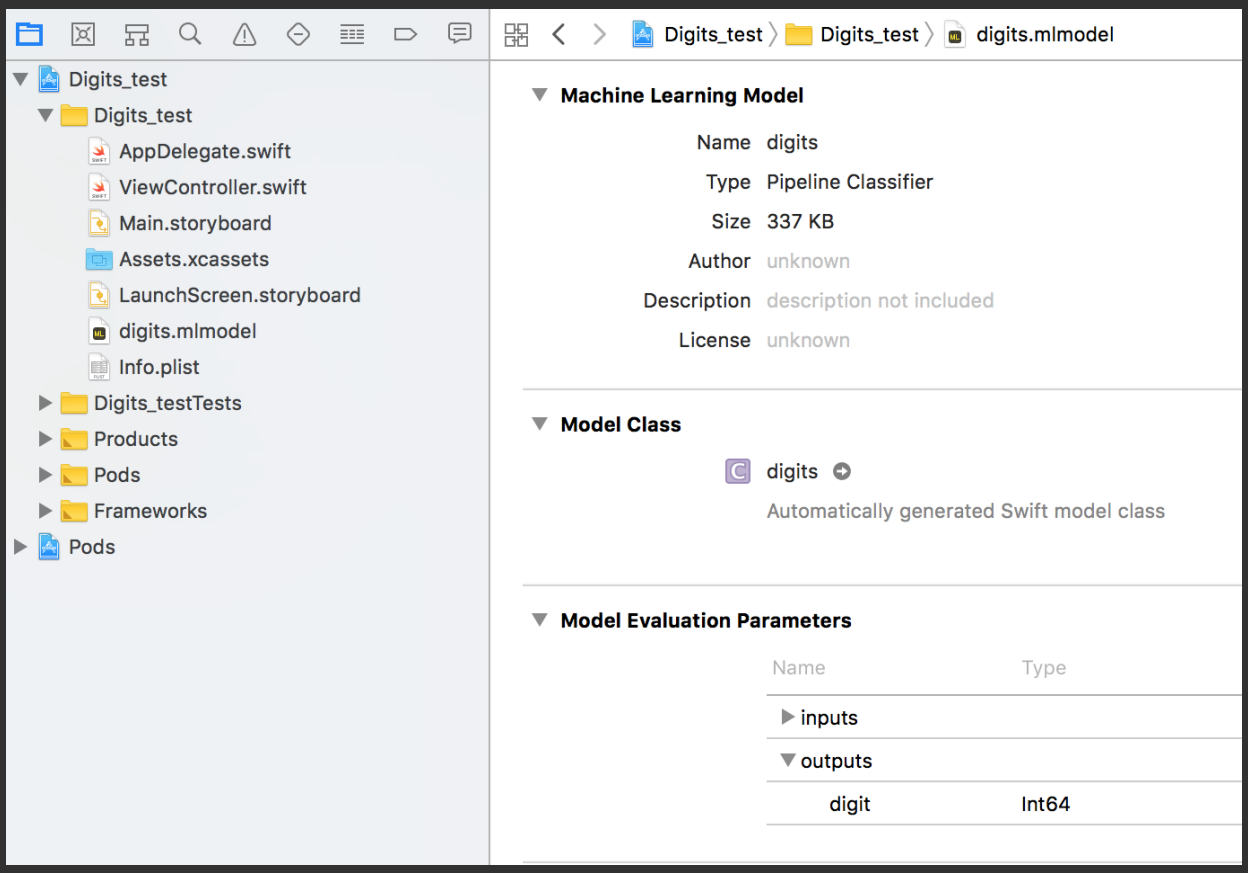


- 检查下Digits.mlmodel文件，大概有400Kb
- 不要在列表中右键保存，这样下载的是一个只有8Kb的文件（我在这个坑上花了不少时间）
- 也可以在**coreml_model.save**的时候直接使用本地路径存储下来（例如 **/Users/[Your-**

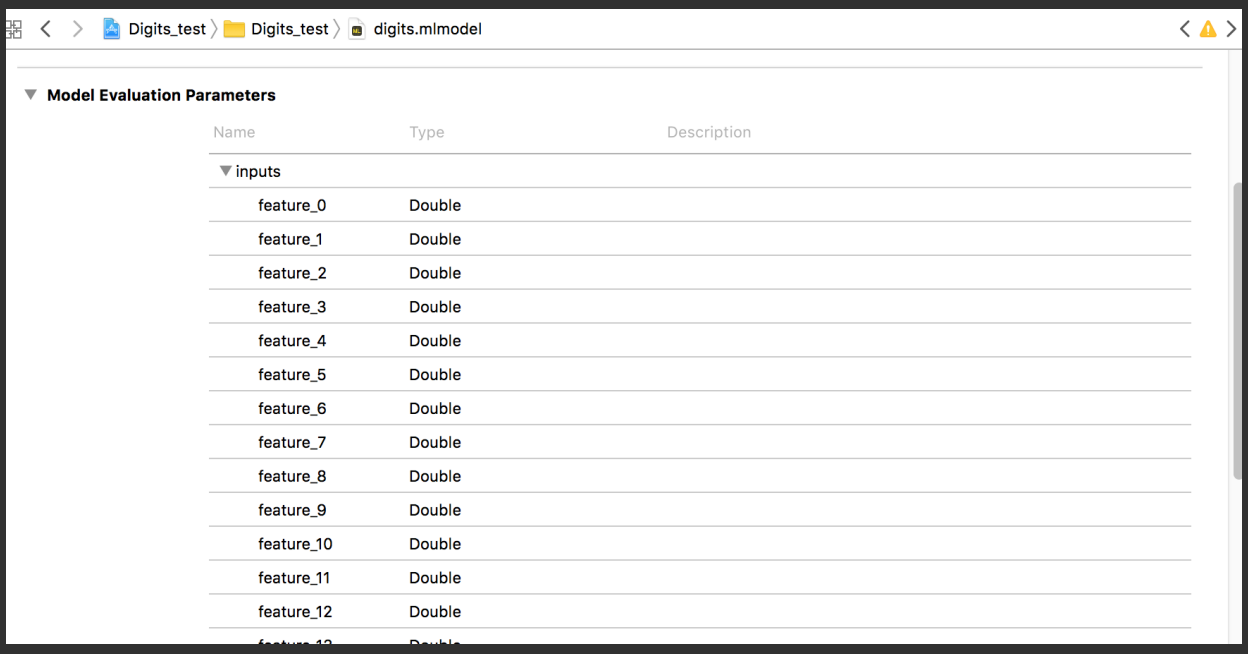
Name]/Downloads/digits.mlmodel)

模型导入

Xcode中新建一个iOS项目，将上一节保存下来的文件拖入到项目中，Xcode会自动解析并列出模型的接口：



我们可以看到模型的输出outputs的是一个Int64类型的64位整型（实际只会输出表示数字识别结果的0-9），展开inputs:



输入是 feture0 - feature_63 总共64个Double的参数，你一定会想：一个函数有64个参数，那写参数名都得写很久啊！

是的，所以自己来想想办法如何把参数优化一下，让使用起来更加方便，目前我们先将就使用。

开始编码

我们需要预测的是一张图片，而输入的参数是64个Double，这64个Double是8*8的点阵图每个点的灰度值，所以要先把图片转换成64个Double，思考一下如果是你会如何处理?

以下是我的想法：

- 先通过摄像头拍照
- 把照片缩小为8*8共64个像素的大小
- 把照片处理成灰度图
- 将每个像素的灰度值转化为一个Double
- 最后就有了64个可供输入的Double

直接上代码（代码较少，全部都写在**ViewController.swift**里面了）：

```
import SwifterSwift // 使用了SwifterSwift来简化颜色的操作

// 从一个UIColor中取出灰度值
// (0.0 - 1.0)
// 0.0 -> 最白 1.0 -> 最黑
extension UIColor {
    func getGray() -> Double {
        let rgb = self.rgbComponents
        // 灰度值的一种算法，red * 0.3 + green * 0.59 + blue * 0.11
        var gray = (
            rgb.red.double * 0.3
            +
            rgb.green.double * 0.59
            +
            rgb.blue.double * 0.11
        ).int
        // 设定与一个阈值，以降低光线或背景的影响
        if gray >= 200 {
            gray = 255
        }

        return (255 - gray.double) / 255
    }
}
```

```
// 从UIImage中取出某个像素点的UIColor
extension UIImage {
    func getPixelColor(pos: CGPoint) -> UIColor? {

        guard let pixelData = self.cgImage?.dataProvider?.data else {
            return nil
        }
        let data: UnsafePointer<UInt8> = CFDataGetBytePtr(pixelData)

        let pixelInfo: Int = ((Int(self.size.width) * Int(pos.y)) +
Int(pos.x)) * 4

        let r = CGFloat(data[pixelInfo]) / CGFloat(255.0)
        let g = CGFloat(data[pixelInfo+1]) / CGFloat(255.0)
        let b = CGFloat(data[pixelInfo+2]) / CGFloat(255.0)
        let a = CGFloat(data[pixelInfo+3]) / CGFloat(255.0)

        return UIColor(red: r, green: g, blue: b, alpha: a)
    }
}
```

```
// 对模型进行扩展，将参数封装，以方便调用
extension digits {
  // 封装模型的预测方法，使用一个8*8的UIImage作为参数，方便调用
  func prediction(image: UIImage) -> Int64? {
        var data: [Double] = []
        for y in 0..<Int(image.size.height) {
            for x in 0..<Int(image.size.width) {
                guard let c = image.getPixelColor(pos: CGPoint(x: x,
y: y)) else {
                    break
                }
                data.append(Double(15 - Int(c.getGray().double/16)))
            }
        }
        return prediction(data: data)
    }
    // 封装一下模型的预测方法，把64个参数封装在一个数组里，方便调用
    func prediction(data: [Double]) -> Int64? {
        guard data.count == 64 else {
            return nil
```

```
 }

if let result = try? prediction(
     feature_0: data[0],
     feature_1: data[1],
     feature_2: data[2],
     feature_3: data[3],
     feature_4: data[4],
     feature_5: data[5],
     feature_6: data[6],
     feature_7: data[7],
     feature_8: data[8],
     feature_9: data[9],
     feature_10: data[10],
     feature_11: data[11],
     feature_12: data[12],
     feature_13: data[13],
     feature_14: data[14],
     feature_15: data[15],
     feature_16: data[16],
     feature_17: data[17],
     feature_18: data[18],
     feature_19: data[19],
     feature_20: data[20],
     feature_21: data[21],
     feature_22: data[22],
     feature_23: data[23],
     feature_24: data[24],
     feature_25: data[25],
     feature_26: data[26],
     feature_27: data[27],
     feature_28: data[28],
     feature_29: data[29],
     feature_30: data[30],
     feature_31: data[31],
     feature_32: data[32],
     feature_33: data[33],
     feature_34: data[34],
     feature_35: data[35],
     feature_36: data[36],
     feature_37: data[37],
     feature_38: data[38],
     feature_39: data[39],
     feature_40: data[40],
     feature_41: data[41],
```

```
            feature_42: data[42],
            feature_43: data[43],
            feature_44: data[44],
            feature_45: data[45],
            feature_46: data[46],
            feature_47: data[47],
            feature_48: data[48],
            feature_49: data[49],
            feature_50: data[50],
            feature_51: data[51],
            feature_52: data[52],
            feature_53: data[53],
            feature_54: data[54],
            feature_55: data[55],
            feature_56: data[56],
            feature_57: data[57],
            feature_58: data[58],
            feature_59: data[59],
            feature_60: data[60],
            feature_61: data[61],
            feature_62: data[62],
            feature_63: data[63]
        ) {
            return result.digit
        }
        return nil
    }
}
```

```
// UI
// 使用一个按钮打开摄像头进行拍照，拍照完成后把图片处理成8*8像素，显示出来，然后用一个Label显示预测结果)
let pickImageButton = UIButton(type: UIButtonType.roundedRect)
let resultLabel = UILabel(text: "no result")
let imageView = UIImageView()

override func viewDidLoad() {
    super.viewDidLoad()

    pickImageButton.frame = CGRect(x: 0, y: 0, width: 100, height: 50)
    pickImageButton.setTitle("Pick Image", for: .normal)
```

```
        view.addSubview(pickImageButton)
        pickImageButton.addTarget(self, action:
    #selector(startPickImage(sender:)), for:
    UIControlEvents.touchUpInside)

        resultLabel.frame = CGRect(x: 300, y: 200, width: 100, height: 50)
        view.addSubview(resultLabel)

        imageView.frame = CGRect(x: 50, y: 300, width: 256, height: 256)
        imageView.contentMode = .scaleAspectFill
        view.addSubview(imageView)
    }
```

```
    // 实现拍照的代理方法
    extension ViewController: UIImagePickerControllerDelegate {
        // pickImageButton Action
        @objc func startPickImage(sender: Any) {
            takePicture()
        }
        // 拍照
        func takePicture() {
            guard UIImagePickerController.isSourceTypeAvailable(.camera)
    else {
                resultLabel.text = "camera not available"
                return
            }

            let cameraPicker = UIImagePickerController()
            cameraPicker.delegate = self
            cameraPicker.sourceType = .camera
            cameraPicker.allowsEditing = false
            present(cameraPicker, animated: true, completion: nil)
        }
        // 拍照取消代理方法
        func imagePickerControllerDidCancel(_ picker:
    UIImagePickerController) {
            dismiss(animated: true, completion: nil)
        }
        // 拍照完成代理方法
        func imagePickerController(_ picker: UIImagePickerController,
    didFinishPickingMediaWithInfo info: [String : Any]) {
```

```
        dismiss(animated: true, completion: nil)

        guard let image = info[UIImagePickerControllerOriginalImage]
as? UIImage else {
            resultLabel.text = "no image"
            return
        }

        // 将图片缩小为8*8
        UIGraphicsBeginImageContext(CGSize(width: 8, height: 8))
        image.draw(in: CGRect(x: 0, y: 0, width: 8, height: 8))
        let newImage = UIGraphicsGetImageFromCurrentImageContext()
        UIGraphicsEndImageContext()

        imageView.image = newImage

        guard let targetImage = newImage else {
            resultLabel.text = "new image is nil"
            return
        }

        let digitsModel = digits()
        guard let result = digitsModel.prediction(image: targetImage)
else {
            resultLabel.text = "predict failed"
            return
        }
        resultLabel.text = "\(result)"

        // 将图像装换为预测数据后显示出来
        for y in 0..<Int(targetImage.size.height) {
            for x in 0..<Int(targetImage.size.width) {
                guard let gray = targetImage.getPixelColor(pos:
CGPoint(x: x, y: y))?.getGray() else {
                    break
                }
                let color = UIColor(
                    red: 255 - (gray * 255).int,
                    green: 255 - (gray * 255).int,
                    blue: 255 - (gray * 255).int
                )
                images[y * 8 + x].backgroundColor = color
            }
```

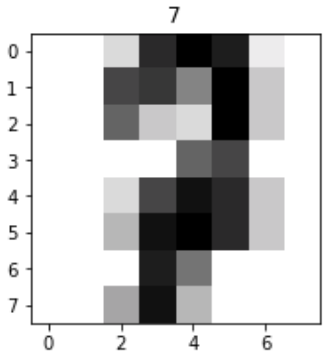
```
            }
        }
    }
```

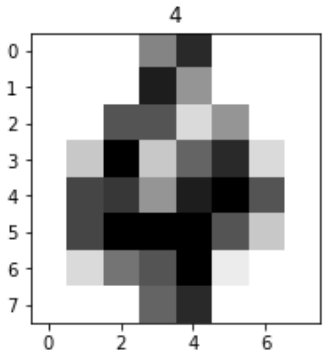
运行结果

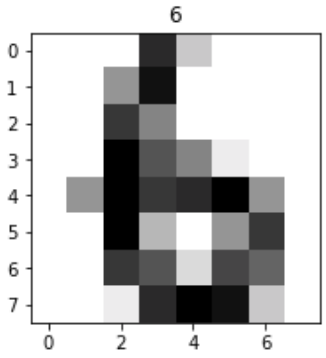
我们可以手写几个数字，然后拍照进行预测。但可能是由于图片处理做得不够好，实际测试下来效果并不理想。所以直接在Jupyter中将测试数据显示在屏幕上，然后使用手机拍照进行测试。

```
show_image(digits.images[1501], digits.target[1501])
show_image(digits.images[1502], digits.target[1502])
show_image(digits.images[1503], digits.target[1503])
```

```
In [42]: show_image(digits.images[1501], digits.target[1501])
         show_image(digits.images[1502], digits.target[1502])
         show_image(digits.images[1503], digits.target[1503])
```







使用App进行拍照预测，可以看到虽然还原出来的图像有一定偏差，但结果还是很准确的：

06:40

Pick Image

7

06:25

Pick Image

4

06:25

Pick Image

6

总结

通过以上学习和练习，相信你已经有种跃跃欲试，想使用机器学习来做点什么的冲动了。希望大家能够保持对机器学习的兴趣与求知欲，充满自信的继续往后的探索学习。

要点回顾

- 目前我们所知道的两种机器学习类别：**预测** 和 **分类**，而有时也会把分类说成是预测。例如上面的App，所谓“预测图片上的数字”，实则是将图片上的数字在0-9中进行分类。能准确判断问题的类别，对于后续的步骤非常重要。
- Python环境的搭建，以及Python2和Python3的区别。
- 我们使用了**sklearn(scikit-learn)**这个机器学习库，它是一个封装好，只需少许代码即可进行模型训练的库。
- 进行训练时所使用的算法是 **SVC（Support Vector Classification）支持向量机分类器**，我们将前2/3的数据用于训练，后1/3的数据进行测试，并输出了测试准确率，以此对模型进行评估。
- 模型的训练需要时间，有的海量数据或密集计算的训练可能耗费很长时间。训练好的模型可以进行保存，以便下次使用。
- 一个训练好的模型可以在多个平台上使用， 但根据平台所支持的格式，可能需要对模型进行转换。

错过了什么

作为一个机器学习的老鸟可能会说，“这篇文章就是标题党，机器学习博大精深，这就算入门啦？！”老鸟说得很对，这篇文章要说入门确实还欠缺不少，不仅跳过了大量原理知识的讲解，还隐去了其中的许多细节：

- 除了sklearn，还有一些其他的常用机器学习库，例如也是使用起来比较简单的**Keras**，Google出品的**TensorFlow**，以及**Theano**，**PyTorch**等。
- 我们训练模型时所使用的数据是现成的，而在实际应用中，还需要**数据收集**，**数据清洗(去重，去错)**，**数据标准化(normalization)**等数据处理的步骤，才能进行接下来的训练。
- 除了**SVC**，机器学习中还有很多其他的算法，每种算法各有优缺点及适用范围。
- **调参**是在实际应用的模型训练中一个不可或缺的步骤，通过对输入参数的调整，观察模型输出的准确率，以此来训练出相对更准确的模型。

那接下来呢

- sklearn中还有两个入门使用率很高的数据集：**波士顿房价预测** 和 **鸢尾花分类**，通过查阅资料，完成App开发，通过输入房屋的信息对房价进行预测，以及通过输入鸢尾花的信息对花进行分类。
- 尝试在工作或个人项目中使用上面所学到的机器学习知识来解决一个简单的问题。
- 自查资料，理解机器学习中更多的常用概念：**过拟合**、**欠拟合**、**监督学习**、**非监督学习**、**损失函数**、**优化函数**等等。
- 学习一些常用的机器学习算法。参考资料：主流机器学习算法简介与其优缺点分析
- 了解深度学习，以及它与机器学习的关系。
- 后续可能会有新的相关机器学习在实际项目中应用的文章以及深度学习的入门文章，你可以订阅antscript.com的RSS或者通过这个链接使用邮件订阅更新。
- 分享或转发这篇文章。
- 如发现错误请不吝指正，请联系邮箱antscript@gmail.com。