阿里云开发者社区
ALIBABA CLOUD DEVELOPER COMMUNITY

阿里云机器学习 PAI-DSW入门指南

PAI-DSW云端IDE揭秘
四大实践场景助你快速上手

手把手教你快速从入门到进阶（内附源码）

评估你的相亲战斗力 | CNN手写识别模型
热狗识别模型 | 半小时验证语音降噪

藏经阁
阿里云开发者电子书系列

目录

作者：贾扬清、林伟、谭锋、黄博远、马渝泽、尹果

序

人工智能（AI）是一个自从计算机被发明开始就存在的一个技术领域。从1956年Marvin Minsky、John McCarthy等人在达特茅斯学院的会议中第一次提出人工智能这个概念开始，AI这个领域的概念、技术和研究经历了非常长足的发展。其中，机器学习是人工智能领域当中最核心也是最广泛应用的一个子领域，旨在通过一系列数学的方法，如统计、概率论、线性代数等方法，设计和分析让计算机可以自动学习的算法。这些算法通过从大量数据中获取规律，来对未知的的数据进行预测和决策。机器学习的算法被广泛地用到计算机视觉、语音、自然语言处理、数据挖掘、搜索、广告、游戏、机器人、金融等各种行业。

随着深度学习的兴起，产业界对于机器学习产生了非常强烈的兴趣，也使得机器学习领域开始迅速地走向工程化和系统化。除了机器学习算法本身的不断创新之外，数据和算力的增加也是不可忽视的：大量的数据，特别是移动互联网的兴起，使机器学习算法得以打破传统数据量的限制；由于GPGPU等高性能处理器开始提供大量的算力，又使得我们能够在可控的时间内（以天为单位甚至更短）进行exaflop级别的算法训练。在这些的综合作用下，工业界开始浮现出大量的机器学习系统创新。以2011年Google Brain，即谷歌大脑为代表，机器学习开始迅速从实验室转向业界。

毫无疑问，深度学习（DeepLearning）是当下最热门的人工智能技术，在智能推荐、图像识别、机器翻译、计算广告、自动驾驶等领域都有突破性的进展和应用。而深度学习的成功很大部分得益于新的计算框架和异构计算硬件，譬如Tensorflow和NVIDIA GPU。

然而，对于算法工程师来说，要搭建这样一套学习和工作的环境不是一件容易的事情：需要一个特定版本的操作系统（最好是Linux），一张或多张GPU卡，安装GPU驱动，安装深度学习计算框架和其依赖的软件包等。在调试深度学习算法的过程中，如果说尝试不同的驱动版本和切换各种版本的软件库还勉强可以接受，那么切换硬件环境，特别是更换GPU卡就伤筋动骨了。

那么，有没有能够一劳永逸解决这些苦恼的方式？阿里云机器学习平台PAI出品的一款云端深度学习开发环境：DSW（Data Science Workshop）试图告诉你，这是个肯定的答案。

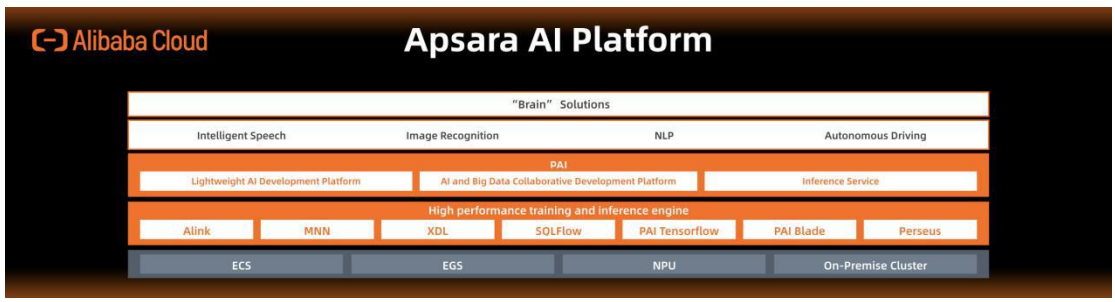
云端IDE：阿里云机器学习与PAI-DSW

经过20年的快速发展，阿里经济体已经组建了一个庞大的商业生态圈，并在支付、云计算、本地生活服务等行业保持互联网巨头地位。2020财年交易额突破1万亿美元，全球第一家；阿里云支撑了2019年双11期间峰值54.4万笔/秒、单日数据处理量达到970PB的世界级的流量洪峰，成为业界第一个实现此壮举的云计算公司。

阿里云机器学习平台正是伴随着这样庞大而复杂的阿里经济体业务成长起来的。下面我们将带着大家掀开阿里云机器学习技术大图的一角，看看阿里云机器学习，特别是机器学习工程上的发展、沉淀和创新。

阿里云机器学习技术大图

我们从用户和技术的两个角度来梳理阿里云机器学习的技术体系大图。从用户的角度来说，根据使用机器学习的深度不同，在云栖大会上，我们展示了飞天AI平台的技术分层关系：



（注：不是一个完整的产品列表，而是一些核心的样例）

从技术的角度说，机器学习从算法到底层的硬件，都涉及到不同的技术方向。下面是我们对于核心技术能力上的一个总体描述：

每个技术方向上都形成各自的布局和沉淀，接下来我们会重点讲述作为机器学习重要组成部分的工程能力体系建设。

阿里云机器学习工程能力体系

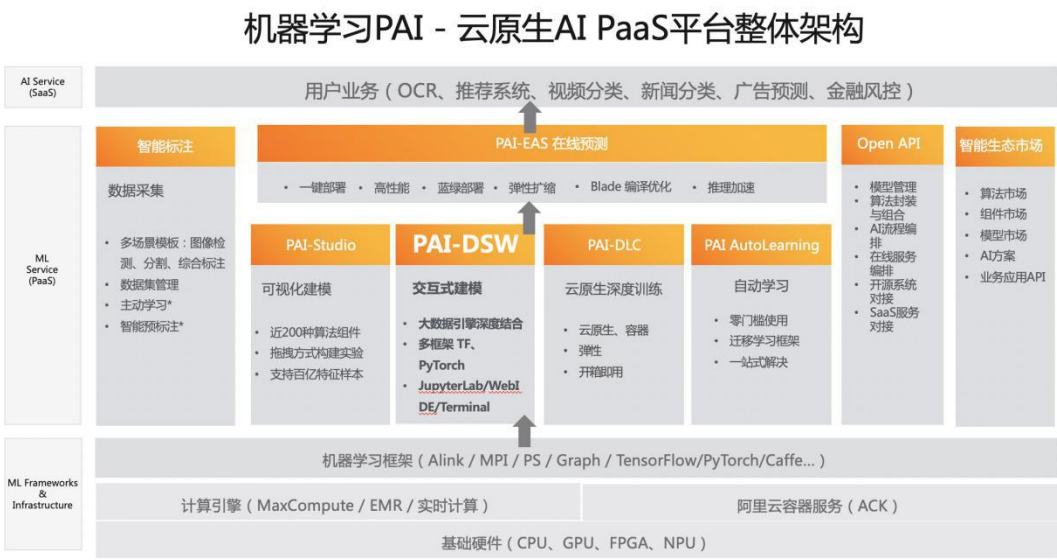
阿里云在机器学习工程体系建设上，也经历了各领域业务需求驱动和技术驱动分阶段螺旋式递进上升的过程。由最初的通过传统机器学习算法进行数据价值的粗加工，到今天以深度学习为主、支撑各类“行业大脑”解决方案的人工智能工程体系。

阿里云的机器学习工程能力体系建设始终围绕着更高效的融合人工智能三要素（算法、数据、算力）进行展开，即追求不断提升整个工程体系中的计算效率、数据效率以及工程效率，从而能够更好的支撑阿里经济体各方面业务快速发展的需求，并通过阿里云对外进行技术输出，推动人工智能领域的技术变革，产生更大的社会效益，实现普惠人工智能。

经过多年的发展创新，阿里云在AI托管平台技术层进行了系统性的建设，极大提升了算法研发、共享、部署、输出的效率，在此基础上沉淀出多个具有用户粘性和场景差异化的开发平台，这里我们选取阿里云机器学习PAI(Platform of Artificial Intelligence)作为代表来着重来介绍。

PAI是一款覆盖机器学习全流程的一站式机器学习平台产品，集数据预处理、特征工程、自动调参、模型训练、在线预测为一体，为用户提供低门槛、高性能的云端机器学习服务。

PAI相关技术脱胎于阿里集团内数十个BU的上千个业务体系，沉淀了大量的覆盖各个领域的优质分布式算法、框架、平台等，同时也在不断完善和扩充机器学习生态。



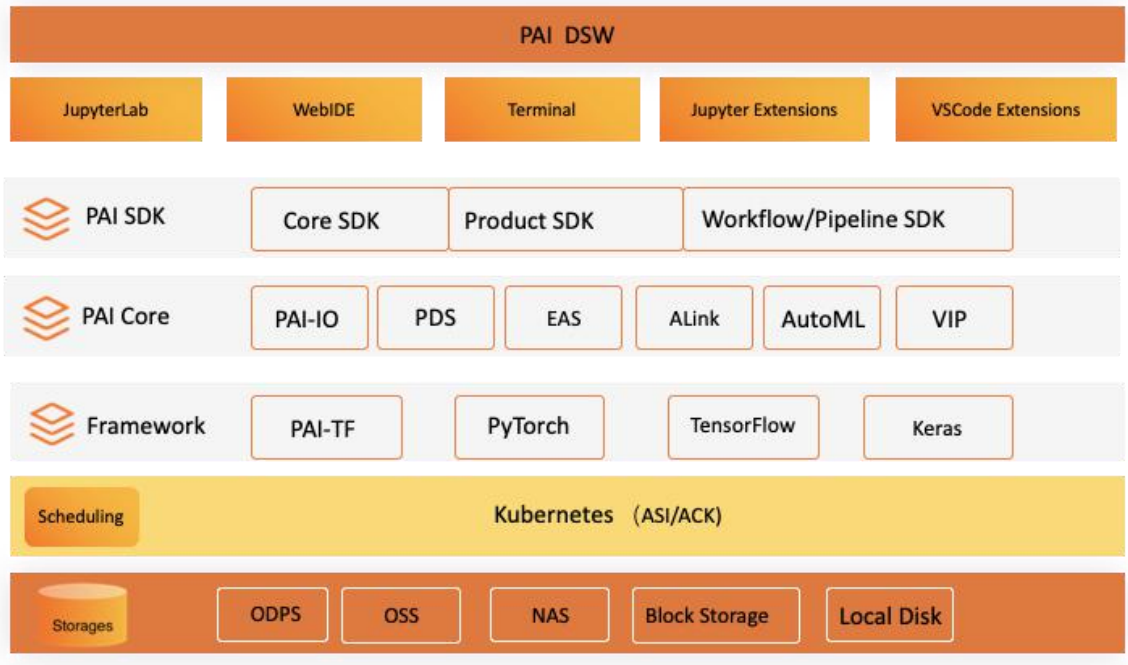
阿里云机器学习PAI-DSW

作为在AI战线上辛勤耕耘的算法工作者，你是否也常常遇到下面的情形：

> 算法需要运行在GPU上，可是长时间申请不到GPU机器，只能干着急。
>
> 终于GPU机器申请到了，却不能马上开始使用，需要先安装GPU驱动和各种依赖等等，感觉有些浪费时间。
>
> 好不容易机器环境弄好了，可当某天更新算法代码后变得很慢，排查半天才发现是GPU驱动需要升级补丁，很是无奈。
>
> 生产环境机器网络隔离，在线上要debug代码，只能使用GDB在命令行进行，开发效率大大降低。
>
> 在本地采用PyCharm这样的IDE开发好代码，而数据在生产环境，不允许下载，只能把代码拷贝到线上机器运行，发现问题后，又得回到本地修改调试后再来一遍，非常不便。
>
> PAI Studio采用图形化拖拽式，像搭积木一样分分钟就构建一个完整的工作流，很炫酷。 但想要定制发布自己的组件时，却不知从何下手。

在长期与算法工程师同学沟通合作的过程中，我们发现了算法工程师的面临的这些问题。提升机器学习工程效率，降低人工智能使用门槛，急需一个简单、轻量、好用的工具平台，让算法工程师更加专注于模型设计本身。PAI DSW（Data Science Workshop）就是PAI团队，为解决算法工程师的以上痛点，新推出的一款云端机器学习开发IDE。

PAI-DSW集成了Jupyterlab、WebIDE等多种开源项目，在阿里巴巴集团内上百个BU和上千名工程师的打磨之下性能和功能上都进行了一定的调优。数据上打通了ODPS等多个数据源，方便用户在构建模型的时候免去重新构建数据管道的工作。同时，在深度学习上，PAI-DSW内置了Tensorboard，可以通过简单的拖拽的方式来帮助深度学习的开发者更好的完成深度学习场景下神经网络的建模。下图展示了DSW在机器学习平台PAI产品架构中的位置：



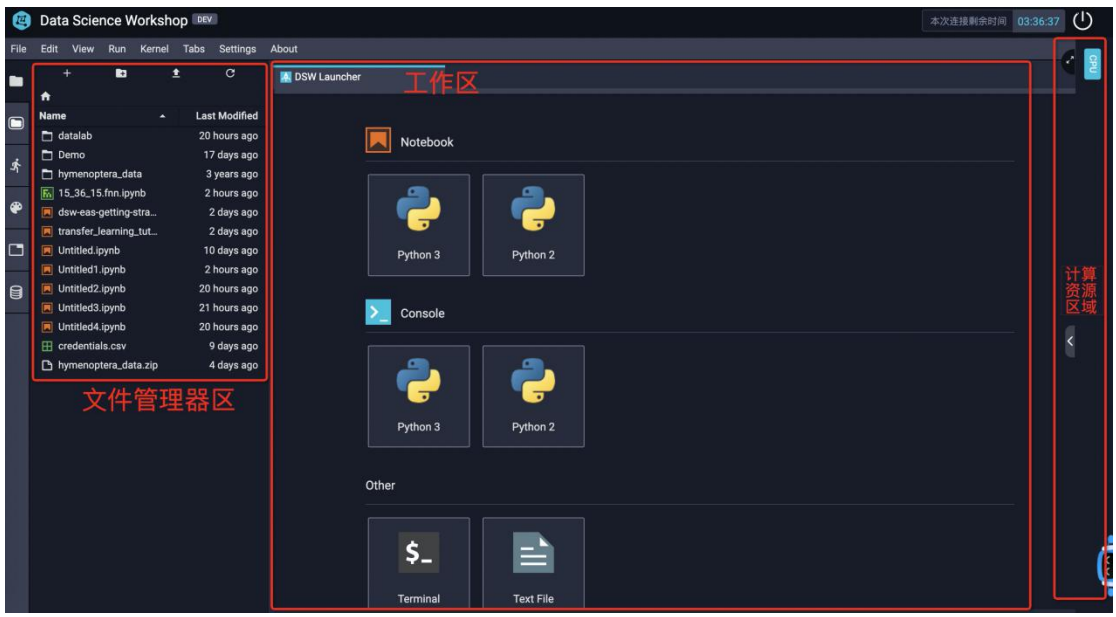
（DSW在机器学习平台PAI产品架构中的位置）

简单来说，PAI-DSW可以实现多实例、多环境，GPU/CPU资源、JupyterLab、WebIDE以及全屏使用Terminal无干扰工作。目前PAI-DSW已经向所有阿里云的用户免费开放了探索者版，只需要登陆阿里云然后打开 https://dsw-dev.data.aliyun.com/#/ 即可即刻开始云上数据科学之旅。本书后面两个章节将详细介绍如何使用PAI-DSW这一简单好用的工具。

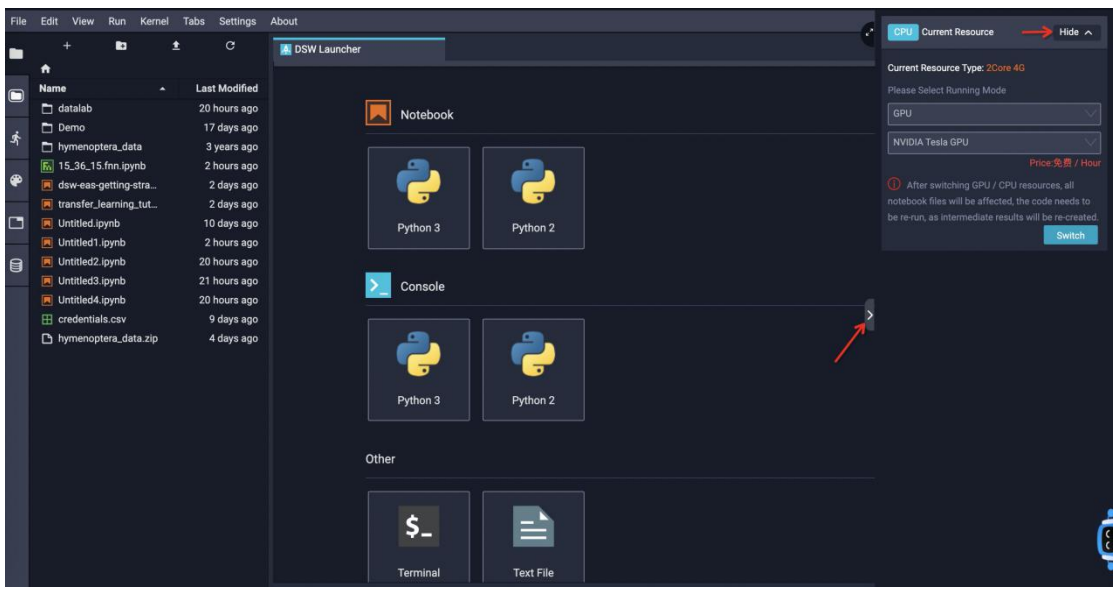
新手上路：PAI-DSW实验室创建攻略

Step 1：创建并打开你的DSW实验室

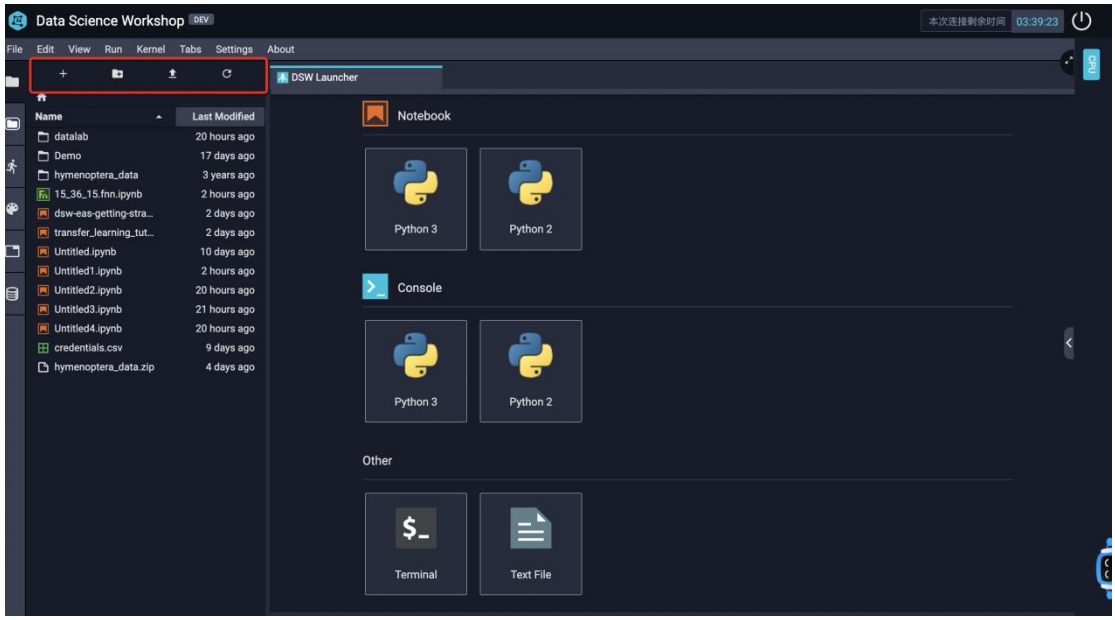
前往https://dsw-dev.data.aliyun.com/#/即可创建并进入你的实验室。在执行这一步之前需要确保已经登陆了阿里云账号和天池账号。进入之后等待几秒后我们会看到如下页面：



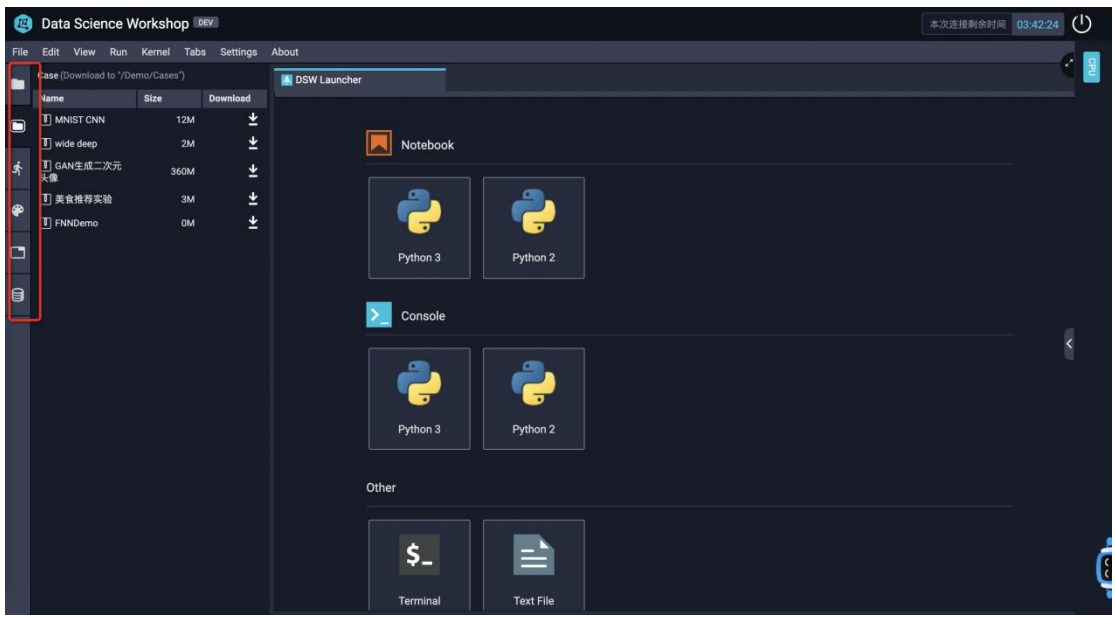
左侧是DSW实验室的文件区，在这里你可以看到在你的实验室里的所有文件夹和文件。双击文件夹即可进如这个文件夹。中间是工作区，所有被打开的文件都会在工作区显示出来。右侧是计算资源区域，在这里你可以看到你当前使用的资源类型。点击右边计算资源区的箭头，即可弹出资源详情，如下图所示：



这里我们也可以点击切换按钮选择我们需要的资源进行切换。

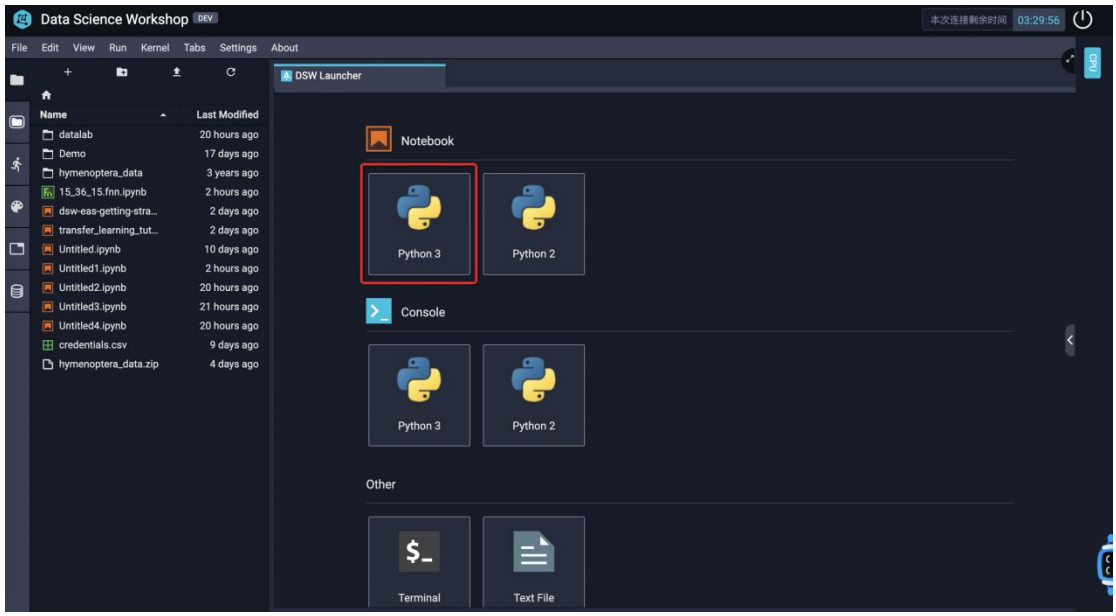


在文件资源管理区的顶部还有4个按钮，从左到右分别对应的是：打开DSW Launcher启动器，新建文件夹，上传文件以及刷新当前文件夹。



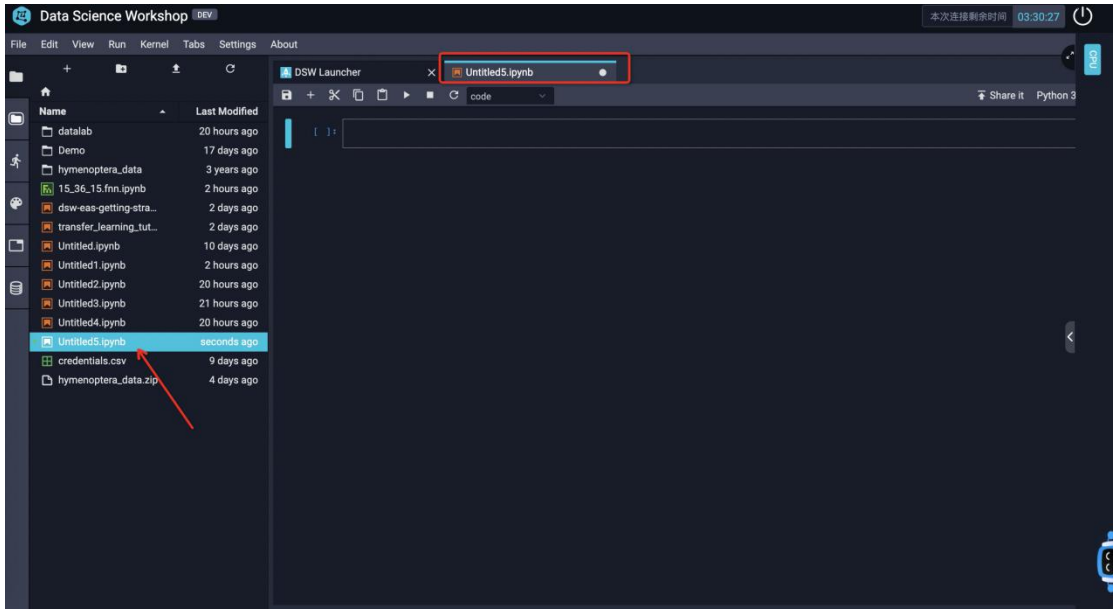
在文件夹左侧还有一栏Tab，每个图标从上到下分别代表了：文件资源管理器，案例代码，正在运行的Notebook，命令面板，在工作区打开的Tab，以及天池的数据搜索引擎。

然后我们回到DSW Launcher启动器,也就是工作区默认打开的界面，然后点击Notebook区域中的Python3，如下图所示：



Step 2：创建一个Notebook

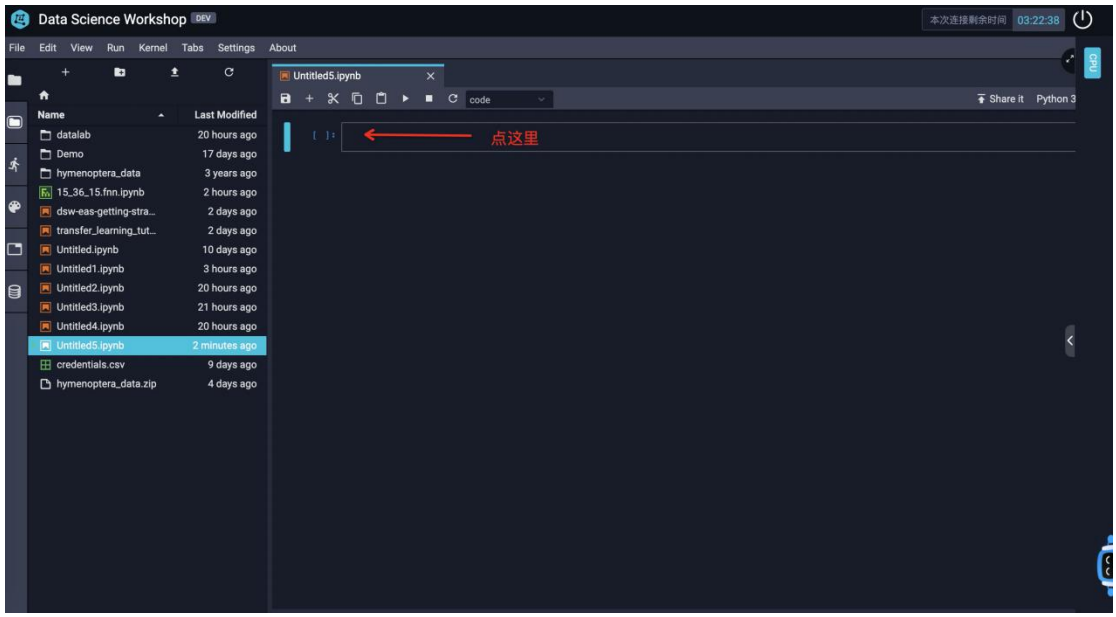
点击了Python3这个图标之后，DSW实验室就会自动为我们创建一个ipynb notebook文件。我们在左侧的资源管理器中也会看到。如下图所示：



这样我们就成功的创建了一个Notebook了。相信熟悉Notebook的你到这一步就很清楚之后怎么操作了～ 如果你以前没有用过也没关系，继续往下看。

Step 3 ：写下你的第一行代码并运行

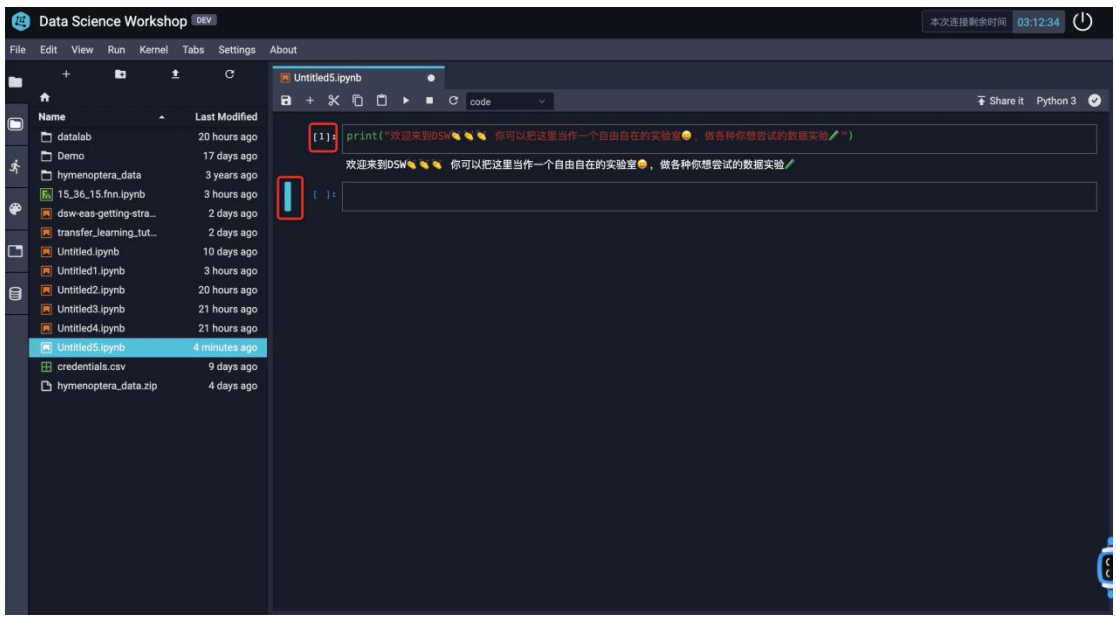
用鼠标点击第一个框框（我们下面以Cell称呼），我们就可以开始从只读模式进入编辑模式开始写代码了。



这里我们可以先输入一个简单的：

```
print("欢迎来到DSW👏👏👏 你可以把这里当作一个自由自在的实验室😁，做各种你想尝试的数据实验🧪")
```

然后我们同时按下 shift+ enter回车这两个键,就可以看到我们的执行结果了，如下图所示：



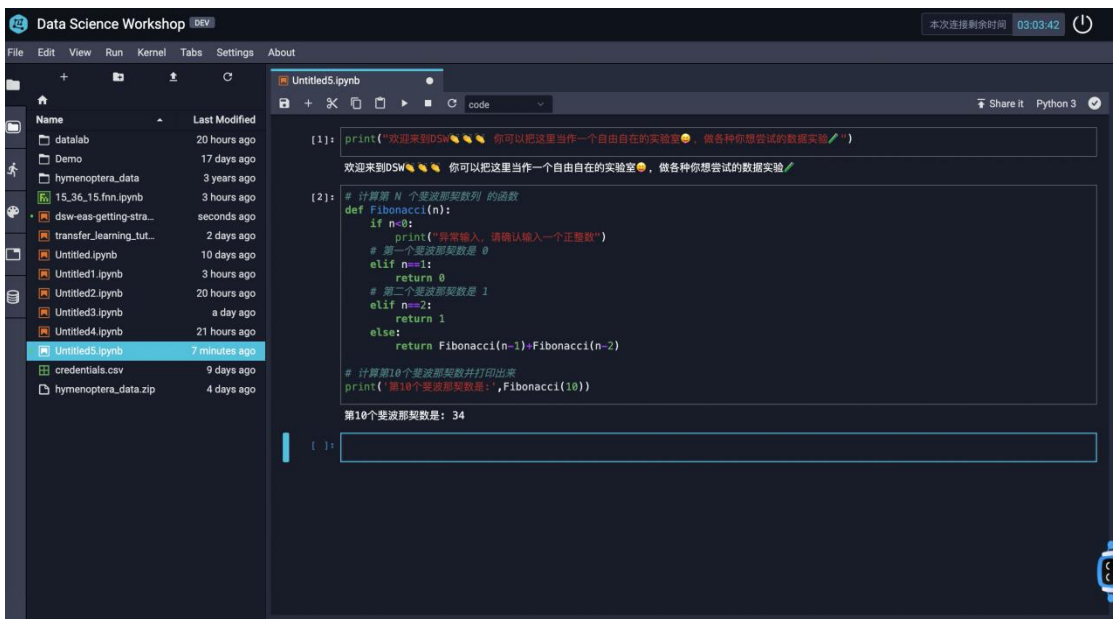
图中标红的数字1表示这个Cell是第一个被执行的，蓝色区域则表示当前聚焦(Focus)的Cell。我们可以在Focus的Cell按下 Enter回车键进入编辑模式。我们也可以按下 Esc键来**退出编辑模式**回到只读模式。在**只读模式**中，我们可以通过方向键上下来切换Focus的Cell。

Step 4：计算一个斐波那契数列

很好，现在你已经熟悉了DSW最基本的运行Cell的方式，接下来就让我们编写一个简单的斐波那契数列计算的函数来计算这个数列的第10项。把下方的代码拷贝到新生成的Cell中即可：

```
# 计算第 N 个斐波那契数列 的函数
def Fibonacci(n):
    if n<0:
        print("异常输入，请确认输入一个正整数")
    # 第一个斐波那契数是 0
    elif n==1:
        return 0
    # 第二个斐波那契数是 1
    elif n==2:
        return 1
    else:
        return Fibonacci(n-1)+Fibonacci(n-2)

# 计算第10个斐波那契数并打印出来
print('第10个斐波那契数是:',Fibonacci(10))
```

同样，输入代码后，我们按下 shift+ enter回车 这两个键,就可以看到我们的执行结果了，如下图所示：



这样，你就掌握了DSW最基本的创建Notebook，便携代码和运行代码的方式。你也可以点击左侧案例的图标来游览一下我们已经为你准备好的样例代码，更好的熟悉DSW的使用方式。

快速进阶：PAI-DSW案例四大实战指南

本着理论与实践相结合的方针，本部分将手把手教你从快速相亲、手写字体识别、热狗识别、语音降噪等四大场景实战参与进来，从入门到快速进阶。实际上，PAI-DSW适用的场景还有很多，等你一起来发掘。

案例一：大数据算命系列之用机器学习评估你的相亲战斗力

"用姓名测试爱情，80%准确率！ 俗话说，名如其人，缘分就是人生的后半生，为了寻找真缘分的大有人在，因此也就有了姓名缘分测试。 您现在是不是也正在心动犹豫，也想要一个属于自己的名字配对缘分测试了呢？那就请您赶紧行动吧！" 以上这段话，你一定已经在很多个微信公众号的尾部，电视节目之间的广告以及奇奇怪怪的小网站上看到过了吧。

你一定很好奇，这个缘分测试背后到底是不是有科学依据支撑的。

今天，作为数据科学老司机的我，虽然不能直接帮你测试你和某个特定的人直接的缘分，但是我们可以借助哥伦比亚大学多年研究相亲找对象的心血，通过几个简单的特征来评估你的相亲战斗力指数。

具体模型的测试页面在这里在正式开始实验之前，我们需要寻找一个简单好用方便上手的工具，这里我推荐一波阿里云的DSW探索者版，它对于个人开发者是免费的，同时还有免费的GPU资源可以使用，同时实验的数据还会免费保存30天。点击这里就可以使用，不需要购买，只要登陆就可以使用。今天我们就会通过这个工具来探索人性的奥秘，走进两性关系的神秘空间嘿嘿嘿。

整个实验的数据收集于一个从2002年到2004年的线下快速相亲的实验。在这个实验中，参与者被要求参加多轮与异性进行的快速相亲，每轮相亲持续4分钟，在4分钟结束后，参与者双方会被询问是否愿意与他们的对象再见面。只有当双方都回答了“是”的时候，这次相亲才算是配对成功。

同时，参与者也会被要求通过以量化的方式从 外观吸引力，真诚度，智商，风趣程度，事业心，兴趣爱好 这六个方向来评估他们的相亲对象。

这个数据集同时也包含了很多参加快速相亲的参与者的其他相关信息，比如地理位置，喜好，对于理想对象的偏好，收入水平，职业以及教育背景等等。关于整个数据集的具体特征描述可以参考这个文件。

本次我们实验的目的主要是为了找出，当一个人在参加快速相亲时，到底会有多高的几率能够遇到自己心动的人并成功牵手。

在我们建模分析探索人性的秘密之前，让我们先读入数据，来看看我们的数据集长什么样。

```
import pandas as pd
df = pd.read_csv('Speed Dating Data.csv', encoding='gbk')
print(df.shape)
```

通过观察，我们不难发现，在这短短的两年中，这个实验的小酒馆经历了8000多场快速相亲的实验。由此我们可以非常轻易的推断出，小酒馆的老板应该赚的盆满钵满（大雾）

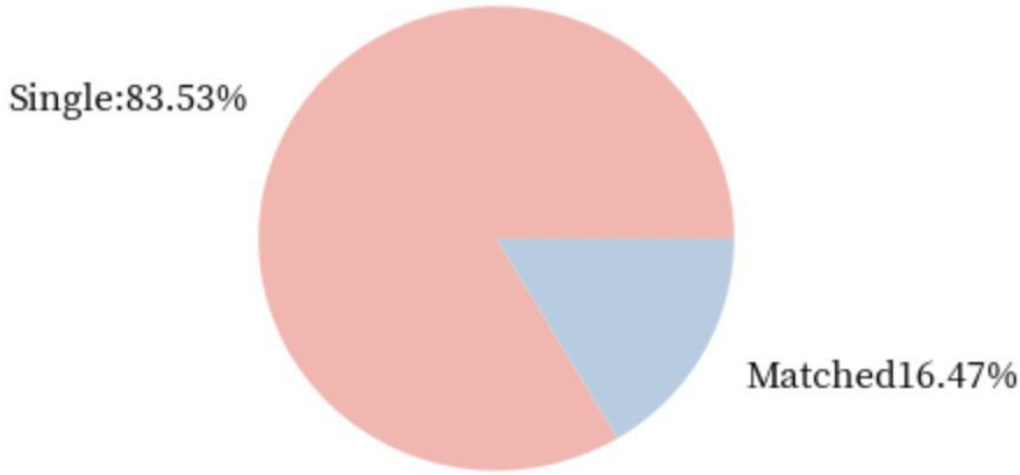
然后从数据的宽度来看，我们会发现一共有接近200个特征。关于每个特征的具体描述大家可以参考这篇文档。然后我们再观察数据的完整度，看看是否有缺失数据。

```
percent_missing = df.isnull().sum() * 100 / len(df)
missing_value_df = pd.DataFrame({
    'column_name': df.columns,
    'percent_missing': percent_missing
})
missing_value_df.sort_values(by='percent_missing')
```

通过以上代码，我们不难发现，其实还有很多的特征是缺失的。这一点在我们后面做分析和建模的时候，都需要关注到。因为一旦一个特征缺失的数据较多，就会导致分析误差变大或者模型过拟合/精度下降。看完数据的完整程度，我们就可以继续往下探索了。

然后第一个问题就来了，在这8000多场的快速相亲中，到底有多少场相亲成功为参加的双方找到了合适的伴侣的？带着这个问题，我们就可以开始我们的第一个探索性数据分析。

```
# 多少人通过Speed Dating找到了对象
plt.subplots(figsize=(3,3), dpi=110,)
# 构造数据
size_of_groups=df.match.value_counts().values

single_percentage = round(size_of_groups[0]/sum(size_of_groups) * 100,2)
matched_percentage = round(size_of_groups[1]/sum(size_of_groups)* 100,2)
names = [
    'Single:' + str(single_percentage) + '%',
    'Matched' + str(matched_percentage) + '%']

# 创建饼图
plt.pie(
    size_of_groups,
    labels=names,
    labeldistance=1.2,
    colors=Pastel1_3.hex_colors
)
plt.show()
```



从上边的饼图我们可以发现，真正通过快速相亲找到对象的比率仅有16.47%。

然后我们就迎来了我们的第二个问题，这个比率和参加的人的性别是否有关呢？这里我们也通过Pandas自带的filter的方式

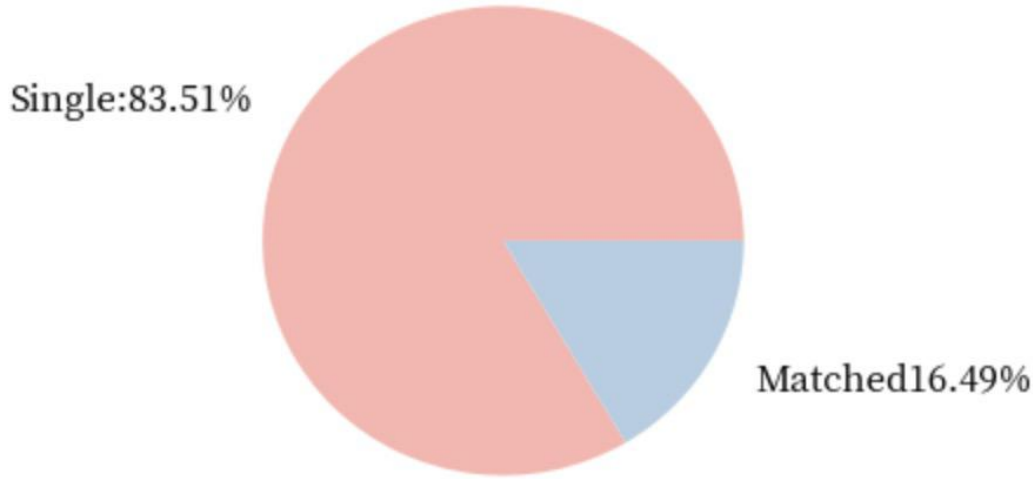
```
df[df.gender == 0]
```

来筛选数据集中的性别。通过阅读数据集的文档，我们知道0代表的是女生，1代表的是男生。然后同理，我们执行类似的代码

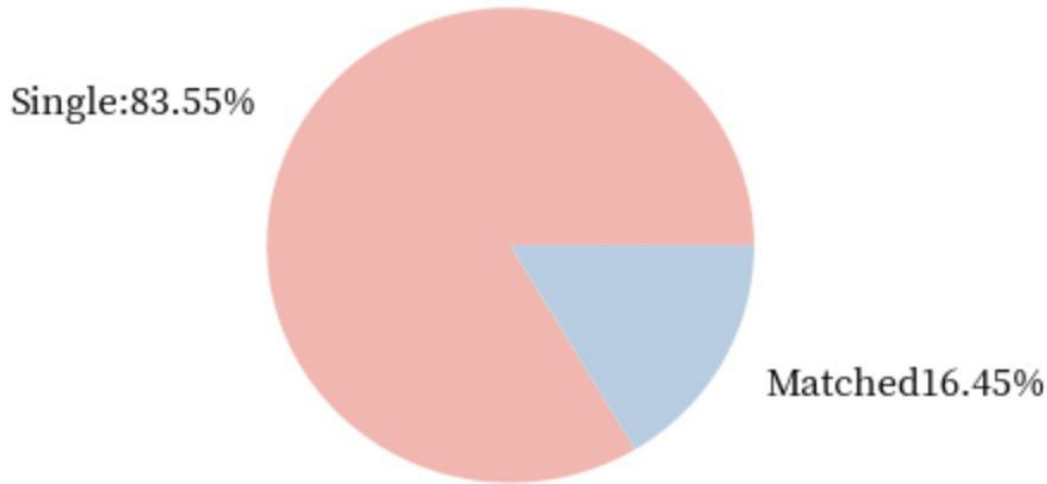
```
# 多少女生通过Speed Dating找到了对象
plt.subplots(figsize=(3,3), dpi=110,)
# 构造数据
size_of_groups=df[df.gender == 0].match.value_counts().values # 男生只需要吧0替换成1即可

single_percentage = round(size_of_groups[0]/sum(size_of_groups) * 100,2)
matched_percentage = round(size_of_groups[1]/sum(size_of_groups)* 100,2)
names = [
  'Single:' + str(single_percentage) + '%',
  'Matched' + str(matched_percentage) + '%']

# 创建饼图
plt.pie(
  size_of_groups,
  labels=names,
  labeldistance=1.2,
  colors=Pastel1_3.hex_colors
)
plt.show()
```

来找出女生和男生分别在快速相亲中找到对象的几率的。

女生的几率：



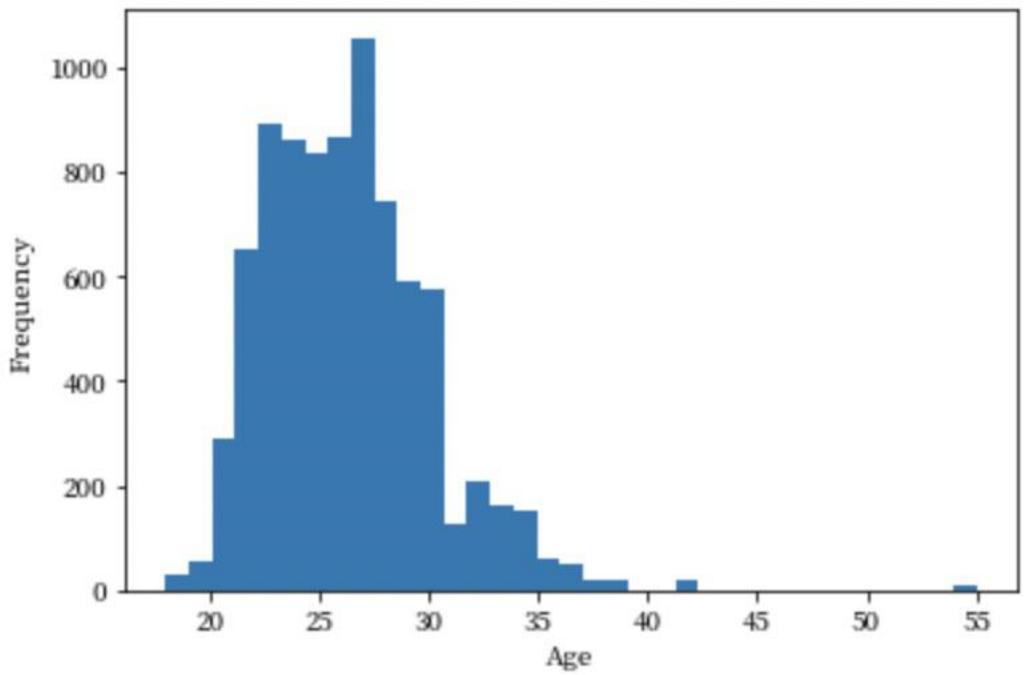
男生的几率：



不难发现，在快速相亲中，女生相比于男生还是稍微占据一些优势的。女生成功匹配的几率比男生成功匹配的几率超出了0.04。

然后第二个问题来了：是什么样的人在参加快速相亲这样的活动呢？真的都是大龄青年（年龄大于30）嘛？这个时候我们就可以通过对参加人群的年龄分布来做一个统计分析。

```
# 年龄分布
age = df[np.isfinite(df['age'])]['age']
plt.hist(age,bins=35)
plt.xlabel('Age')
plt.ylabel('Frequency')
```

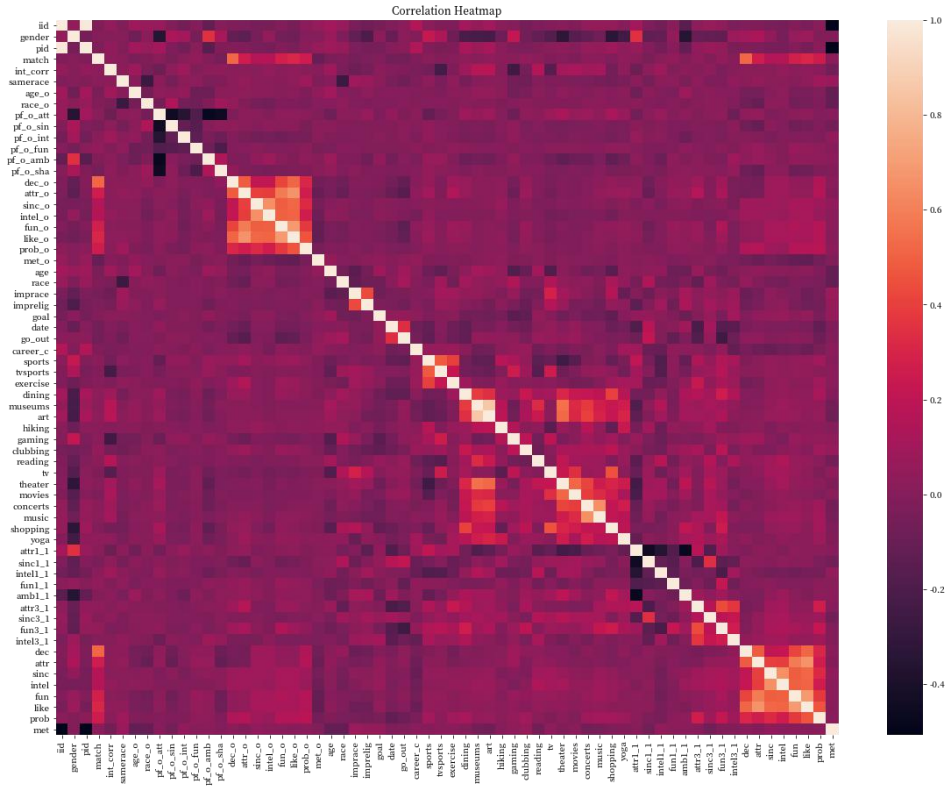


不难发现，参加快速相亲的人群主要是22～28岁的群体。这点与我们的预期有些不太符合，因为主流人群并不是大龄青年。接下来的问题就是，年龄是否会影响相亲的成功率呢？和性别相比，哪个对于成功率的影响更大？这两个问题在本文就先埋下一个伏笔，不一一探索了，希望阅读文章的你能够自己探索。

但是这里可以给出一个非常好用的探索相关性的方式叫做数据相关性分析。通过阅读数据集的描述，我已经为大家选择好了一些合适的特征去进行相关性分析。这里合适的定义是指：1. 数据为数字类型，而不是字符串等无法量化的值。2.数据的缺失比率较低

```
date_df = df[[
   'iid', 'gender', 'pid', 'match', 'int_corr', 'samerace', 'age_o',
      'race_o', 'pf_o_att', 'pf_o_sin', 'pf_o_int', 'pf_o_fun', 'pf_o_amb',
      'pf_o_sha', 'dec_o', 'attr_o', 'sinc_o', 'intel_o', 'fun_o', 'like_o',
      'prob_o', 'met_o', 'age', 'race', 'imprace', 'imprelig', 'goal', 'date',
      'go_out', 'career_c', 'sports', 'tvsports', 'exercise', 'dining',
      'museums', 'art', 'hiking', 'gaming', 'clubbing', 'reading', 'tv',
      'theater', 'movies', 'concerts', 'music', 'shopping', 'yoga', 'attr1_1',
      'sinc1_1', 'intel1_1', 'fun1_1', 'amb1_1', 'attr3_1', 'sinc3_1',
      'fun3_1', 'intel3_1', 'dec', 'attr', 'sinc', 'intel', 'fun', 'like',
      'prob', 'met'
]]

# heatmap
plt.subplots(figsize=(20,15))
```

```
ax = plt.axes()
ax.set_title("Correlation Heatmap")
corr = date_df.corr()
sns.heatmap(corr,
        xticklabels=corr.columns.values,
        yticklabels=corr.columns.values)
```



通过上面这张图这张相关性分析的热力图，我们可以先关注一些特别亮的和特别暗的点。比如我们可以发现，在 pf_o_att这个表示相亲对象给出的外观吸引力这个特征上，和其他相亲对象给出的评分基本都是严重负相关的，除了pf_o_fun这一特征。由此我们可以推断出两个点：

1. 大家会认为外观更加吸引人的人在智商，事业心，真诚度上表现会相对较差。换句话说，可能就是颜值越高越浪
2. 幽默风趣的人更容易让人觉得外观上有吸引力,比如下面这位幽默风趣的男士(大雾):

然后我们再看看我们最关注的特征 match，和这一个特征相关性比较高的特征是哪几个呢？不难发现，其实就是'attr_o','sinc_o','intel_o','fun_o','amb_o','shar_o'这几个特征，分别是相亲对方给出的关于外观，真诚度，智商，风趣程度，事业线以及兴趣爱好的打分。接下来我们就可以根据这个来进行建模了。首先我们将我们的特征和结果列都放到一个Dataframe中，然后再去除含有空值的纪录。 最后我们再分为X和Y用来做训练。当然分为X，y之后，由于我们在最开始就发现只有16.47%的参与场次中成功匹配了，所以我们的数据有严重的不均衡，这里我们可以用SVMSMOTE来增加一下我们的数据量避免模型出现过度拟合。

```
# preparing the data
clean_df = df[['attr_o','sinc_o','intel_o','fun_o','amb_o','shar_o','match']]
clean_df.dropna(inplace=True)
X=clean_df[['attr_o','sinc_o','intel_o','fun_o','amb_o','shar_o',]]
y=clean_df['match']

oversample = imblearn.over_sampling.SVMSMOTE()
X, y = oversample.fit_resample(X, y)

# 做训练集和测试集分割
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0,
stratify=y)
```

数据准备好之后，我们就可以进行模型的构建和训练了。通过以下代码，我们可以构建一个简单的逻辑回归的模型，并在测试集上来测试。

```
# logistic regression classification model
model = LogisticRegression(C=1, random_state=0)
lrc = model.fit(X_train, y_train)
predict_train_lrc = lrc.predict(X_train)
predict_test_lrc = lrc.predict(X_test)
print('Training Accuracy:', metrics.accuracy_score(y_train, predict_train_lrc))
print('Validation Accuracy:', metrics.accuracy_score(y_test, predict_test_lrc))
```

```
[38]: # logistic regression classification model
      model = LogisticRegression(C=1, random_state=0)
      lrc = model.fit(X_train, y_train)
      predict_train_lrc = lrc.predict(X_train)
      predict_test_lrc = lrc.predict(X_test)
      print('Training Accuracy:', metrics.accuracy_score(y_train, predict_train_lrc))
      print('Validation Accuracy:', metrics.accuracy_score(y_test, predict_test_lrc))

      Training Accuracy: 0.7299097550494199
      Validation Accuracy: 0.7362542955326461
```

我们可以看到结果为0.83左右，这样我们就完成了一个预测在快速相亲中是否能够成功配对的机器学习模型。针对这个模型，数据科学老司机我还专门制作了一个小游戏页面，来测试你的相亲战斗力指数。 同时也欢迎你加入我们的DSW用户交流群，和我们一起交流/探索更多好玩又实用的机器学习/深度学习案例。

案例二：四步训练出自己的CNN手写识别模型

虽然已经 9102 年了MNIST手写数据集也早已经被各路神仙玩出了各种花样，比如其中比较秀的有用MINST训练手写日语字体的。但是目前还是很少有整体的将训练完之后的结果部署为一个可使用的服务的。大多数还是停留在最终Print出一个Accuracy。

这一次，借助阿里云的PAI-DSW来快速构建训练一个手写模型并且部署出一个生产可用级别的服务的教程让大家可以在其他的产品中调用这个服务作出更加有意思的项目。

这篇文章里我们先讲讲如何构建训练并导出这个手写字体识别的模型。整个教程的代码基于Snapchat的ML大佬 Aymeric Damien 的Tensorflow 入门教程系列。

第一步: 下载代码

首先我们可以把代码Clone到本地或者直接Clone到DSW的实例。如何Clone到DSW实例的方法可以参考我的这篇文章。Clone完代码之后我们还需要准备训练所需要的数据集这边可以直接从Yann Lecun的网站下载。我这边然后我们可先运行一遍看一下效果。

```
        sess.run(accuracy, feed_dict={X: mnist.test.images[:256],
                                      Y: mnist.test.labels[:256],
                                      keep_prob: 1.0}))
```

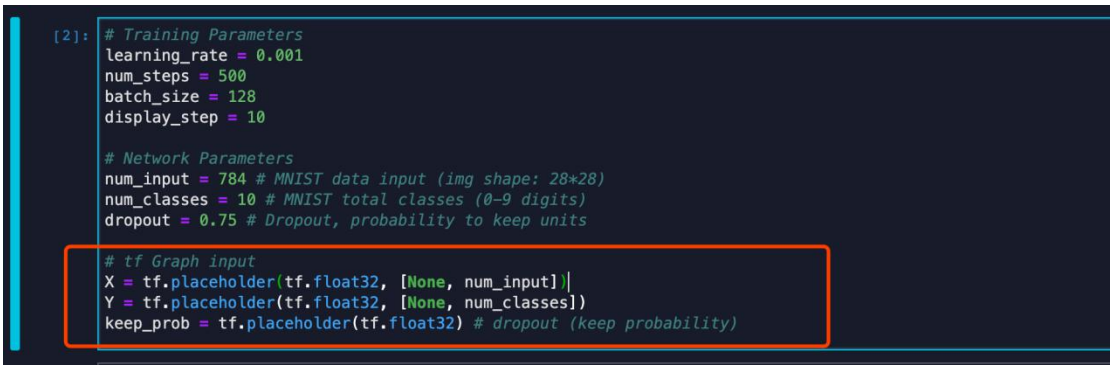
```
Step 1, Minibatch Loss= 47837.0742, Training Accuracy= 0.086
Step 10, Minibatch Loss= 16704.7188, Training Accuracy= 0.289
Step 20, Minibatch Loss= 10984.0859, Training Accuracy= 0.609
Step 30, Minibatch Loss= 3326.6958, Training Accuracy= 0.750
Step 40, Minibatch Loss= 2857.9272, Training Accuracy= 0.812
Step 50, Minibatch Loss= 3253.6921, Training Accuracy= 0.836
Step 60, Minibatch Loss= 3823.8716, Training Accuracy= 0.820
Step 70, Minibatch Loss= 2945.3276, Training Accuracy= 0.844
Step 80, Minibatch Loss= 2441.2561, Training Accuracy= 0.828
Step 90, Minibatch Loss= 1390.5857, Training Accuracy= 0.906
Step 100, Minibatch Loss= 837.2078, Training Accuracy= 0.883
Step 110, Minibatch Loss= 1972.9619, Training Accuracy= 0.938
Step 120, Minibatch Loss= 1829.7719, Training Accuracy= 0.906
Step 130, Minibatch Loss= 1463.5245, Training Accuracy= 0.906
Step 140, Minibatch Loss= 2409.1350, Training Accuracy= 0.906
Step 150, Minibatch Loss= 1615.2670, Training Accuracy= 0.922
Step 160, Minibatch Loss= 2411.8921, Training Accuracy= 0.883
Step 170, Minibatch Loss= 2330.6833, Training Accuracy= 0.867
Step 180, Minibatch Loss= 2049.8115, Training Accuracy= 0.867
Step 190, Minibatch Loss= 406.3709, Training Accuracy= 0.938
Step 200, Minibatch Loss= 1683.3142, Training Accuracy= 0.938
Step 210, Minibatch Loss= 1124.9478, Training Accuracy= 0.938
Step 220, Minibatch Loss= 1093.2625, Training Accuracy= 0.938
Step 230, Minibatch Loss= 1200.0924, Training Accuracy= 0.938
Step 240, Minibatch Loss= 266.2341, Training Accuracy= 0.969
Step 250, Minibatch Loss= 1429.6007, Training Accuracy= 0.906
Step 260, Minibatch Loss= 1073.6968, Training Accuracy= 0.938
Step 270, Minibatch Loss= 1034.2772, Training Accuracy= 0.945
Step 280, Minibatch Loss= 124.0369, Training Accuracy= 0.984
Step 290, Minibatch Loss= 670.6676, Training Accuracy= 0.945
Step 300, Minibatch Loss= 969.6609, Training Accuracy= 0.938
Step 310, Minibatch Loss= 832.7277, Training Accuracy= 0.945
Step 320, Minibatch Loss= 547.6849, Training Accuracy= 0.945
Step 330, Minibatch Loss= 705.9149, Training Accuracy= 0.953
Step 340, Minibatch Loss= 550.0765, Training Accuracy= 0.953
Step 350, Minibatch Loss= 804.8413, Training Accuracy= 0.938
Step 360, Minibatch Loss= 767.2852, Training Accuracy= 0.953
Step 370, Minibatch Loss= 713.3838, Training Accuracy= 0.953
Step 380, Minibatch Loss= 1196.2185, Training Accuracy= 0.953
Step 390, Minibatch Loss= 815.1002, Training Accuracy= 0.945
Step 400, Minibatch Loss= 229.9126, Training Accuracy= 0.984
Step 410, Minibatch Loss= 492.2887, Training Accuracy= 0.953
Step 420, Minibatch Loss= 553.7189, Training Accuracy= 0.969
Step 430, Minibatch Loss= 596.3711, Training Accuracy= 0.969
Step 440, Minibatch Loss= 216.9878, Training Accuracy= 0.969
Step 450, Minibatch Loss= 723.7683, Training Accuracy= 0.906
Step 460, Minibatch Loss= 724.8499, Training Accuracy= 0.969
Step 470, Minibatch Loss= 725.8353, Training Accuracy= 0.945
Step 480, Minibatch Loss= 1194.0928, Training Accuracy= 0.938
Step 490, Minibatch Loss= 1449.9185, Training Accuracy= 0.914
Step 500, Minibatch Loss= 245.0169, Training Accuracy= 0.969
Optimization Finished!
Testing Accuracy: 0.97265625
```

```
[ ]:
```

我们可以看到代码Clone下来之后直接运行就已经帮我们训练出了model并且给出了现在这个Model的精度。在500个batch之后准确率达到了95%以上而且基于GPU的DSW实例训练这500个Batch只需要十几秒的时间。

第二步: 修改部分代码使得可以自动导出SavedModel

这一步就是比较重要的地方了我们第一个需要关注的就是当前的这个Model里面的Input和Output. Input还比较清楚我们直接找所有placeholder就可以了。



```
[2]: # Training Parameters
     learning_rate = 0.001
     num_steps = 500
     batch_size = 128
     display_step = 10

     # Network Parameters
     num_input = 784 # MNIST data input (img shape: 28*28)
     num_classes = 10 # MNIST total classes (0-9 digits)
     dropout = 0.75 # Dropout, probability to keep units

     # tf Graph input
     X = tf.placeholder(tf.float32, [None, num_input])
     Y = tf.placeholder(tf.float32, [None, num_classes])
     keep_prob = tf.placeholder(tf.float32) # dropout (keep probability)
```

Output这一块就比较复杂了,在当前的model里我们可以看到output并不是直接定义的Y而是softmax之后的prediction

```
# Construct model
logits = conv_net(X, weights, biases, keep_prob)
prediction = tf.nn.softmax(logits)
```

找到了这些之后就比较简单了。首先我们创建一个 Saver , 它可以帮助我们保存所有的tf变量以便之后导出模型使用

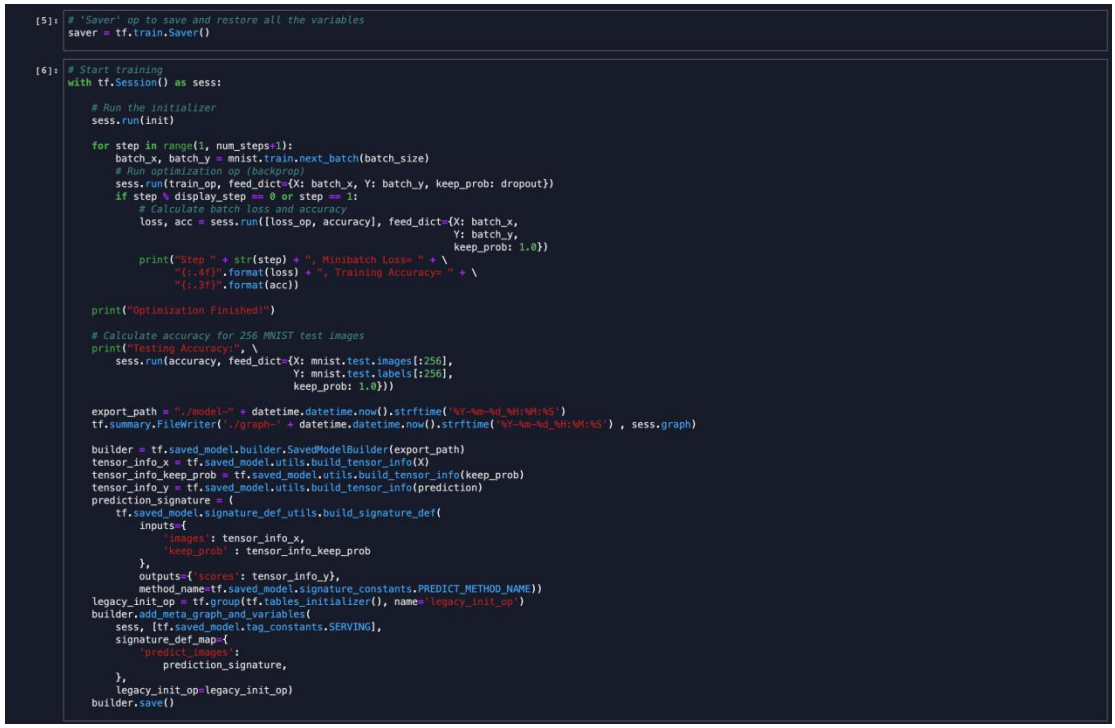
```
# 'Saver' op to save and restore all the variables
saver = tf.train.Saver()
```

然后我们在模型训练的session结束的时候导出模型就好了。我们可以通过以下这段代码来导出我们训练好的模型

```
import datetime
# 声明导出模型路径这边加入了时间作为路径名 这样每次训练的时候就可以保存多个版本的模型了
export_path = "./model-" + datetime.datetime.now().strftime('%Y-%m-%d_%H:%M:%S')

# 保存训练的日志文件方便如果出问题了我们可以用 tensorboard 来可视化神经网络排查问题
tf.summary.FileWriter('./graph-' + datetime.datetime.now().strftime('%Y-%m-%d_%H:%M:%S') , sess.graph)
```

```
# 构建我们的Builder
builder = tf.saved_model.builder.SavedModelBuilder(export_path)

# 声明各种输入这里有一个X和一个keep_prob作为输入然后
tensor_info_x = tf.saved_model.utils.build_tensor_info(X)
tensor_info_keep_prob = tf.saved_model.utils.build_tensor_info(keep_prob)
tensor_info_y = tf.saved_model.utils.build_tensor_info(prediction)

prediction_signature = (
  tf.saved_model.signature_def_utils.build_signature_def(
    # 声明输入

    inputs={
      'images': tensor_info_x,
      'keep_prob' : tensor_info_keep_prob
    },
    # 声明输出
    outputs={
      'scores': tensor_info_y
    },
    method_name=tf.saved_model.signature_constants.PREDICT_METHOD_NAME
  )
)



legacy_init_op = tf.group(tf.tables_initializer(), name='legacy_init_op')
builder.add_meta_graph_and_variables(
  sess, [tf.saved_model.tag_constants.SERVING],
  signature_def_map={
    'predict_images':
      prediction_signature,
  },
  legacy_init_op=legacy_init_op)
# 保存模型
builder.save()
```

我们可以把这段代码插在这里这样训练完成的时候就会自动导出了



```
[5]: # 'Saver' op to save and restore all the variables
     saver = tf.train.Saver()

[6]: # Start training
     with tf.Session() as sess:

         # Run the initializer
         sess.run(init)

         for step in range(1, num_steps+1):
             batch_x, batch_y = mnist.train.next_batch(batch_size)
             # Run optimization op (backprop)
             sess.run(train_op, feed_dict={X: batch_x, Y: batch_y, keep_prob: dropout})
             if step % display_step == 0 or step == 1:
                 # Calculate batch loss and accuracy
                 loss, acc = sess.run([loss_op, accuracy], feed_dict={X: batch_x,
                                                                      Y: batch_y,
                                                                      keep_prob: 1.0})
                 print("Step " + str(step) + ", Minibatch Loss= " + \
                       "{:.4f}".format(loss) + ", Training Accuracy= " + \
                       "{:.3f}".format(acc))

         print("Optimization Finished!")

         # Calculate accuracy for 256 MNIST test images
         print("Testing Accuracy:", \
             sess.run(accuracy, feed_dict={X: mnist.test.images[:256],
                                           Y: mnist.test.labels[:256],
                                           keep_prob: 1.0}))

         export_path = "./model-" + datetime.datetime.now().strftime('%Y-%m-%d_%H:%M:%S')
         tf.summary.FileWriter('./graph-' + datetime.datetime.now().strftime('%Y-%m-%d_%H:%M:%S') , sess.graph)

         builder = tf.saved_model.builder.SavedModelBuilder(export_path)
         tensor_info_x = tf.saved_model.utils.build_tensor_info(X)
         tensor_info_keep_prob = tf.saved_model.utils.build_tensor_info(keep_prob)
         tensor_info_y = tf.saved_model.utils.build_tensor_info(prediction)
         prediction_signature = (
             tf.saved_model.signature_def_utils.build_signature_def(
                 inputs={
                     'images': tensor_info_x,
                     'keep_prob' : tensor_info_keep_prob
                 },
                 outputs={'scores': tensor_info_y},
                 method_name=tf.saved_model.signature_constants.PREDICT_METHOD_NAME))
         legacy_init_op = tf.group(tf.tables_initializer(), name='legacy_init_op')
         builder.add_meta_graph_and_variables(
             sess, [tf.saved_model.tag_constants.SERVING],
             signature_def_map={
                 'predict_images':
                     prediction_signature,
             },
             legacy_init_op=legacy_init_op)
         builder.save()
```

导出之后应该会有如下的文件结构我们也可以在左边的文件管理器中查看。

```
./model-2019-05-20_13:50:26
├── saved_model.pb
└── variables
    ├── variables.data-00000-of-00001
    └── variables.index

1 directory, 3 files
```

第三步: 部署我们的模型

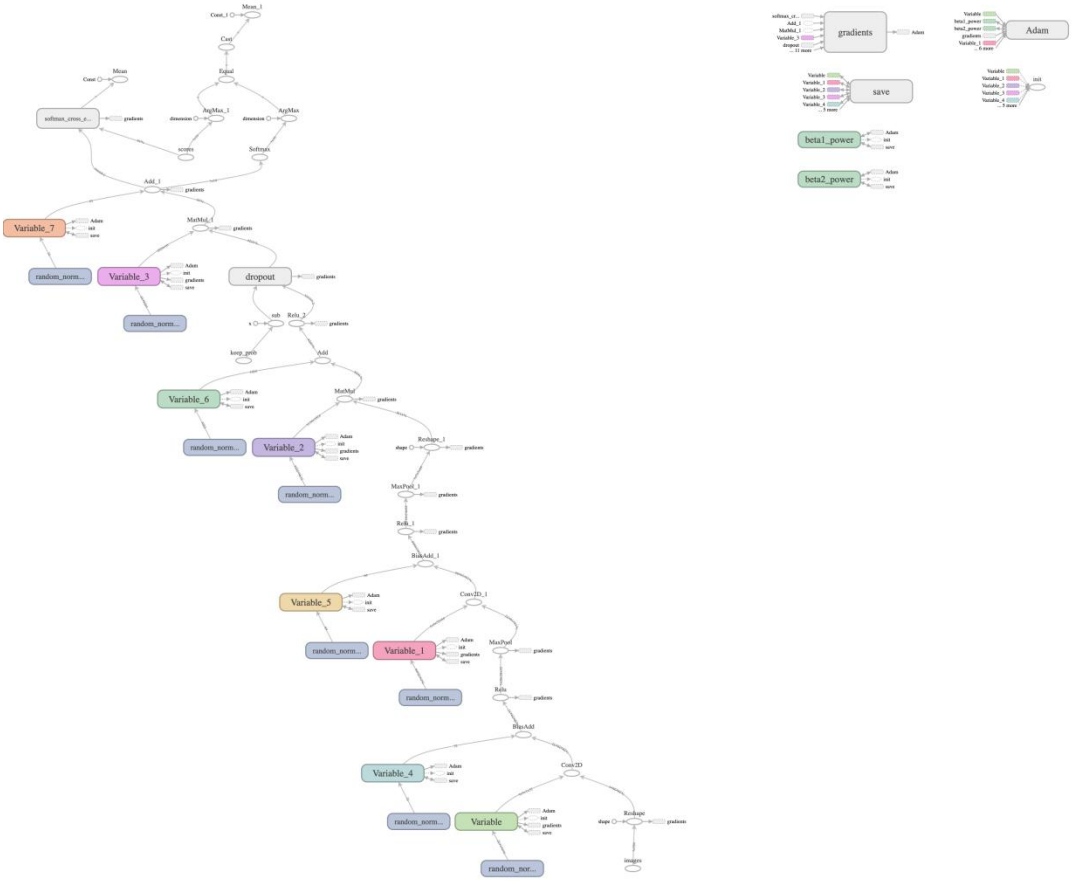
终于到了可以部署的阶段了。但是在部署之前先别那么着急建议用 tensorboard 把训练日志下载到本地之后看一下。

这一步除了可以可视化的解释我们的模型之外还可以帮助我们理清我们的模型的输入和输出分别是什么。

这边我先在有日志文件的路径打开一个tensorboard 通过这个命令

```
tensorboard --logdir ./
```

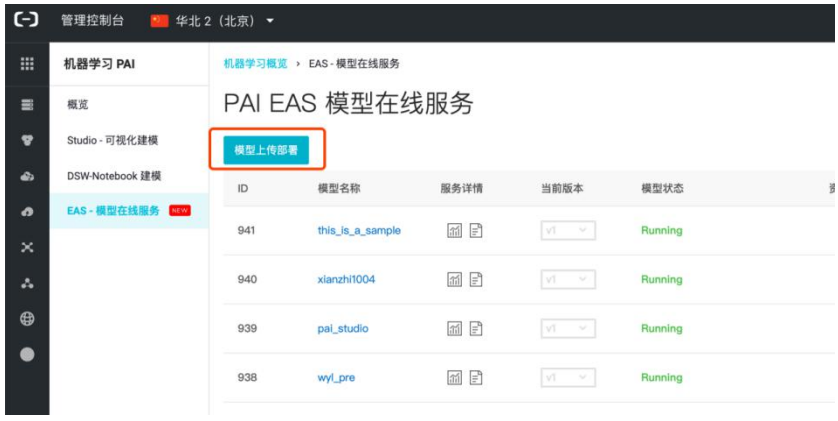
然后我们在游览器里输入默认地址 localhost:6006 就可以看到了。



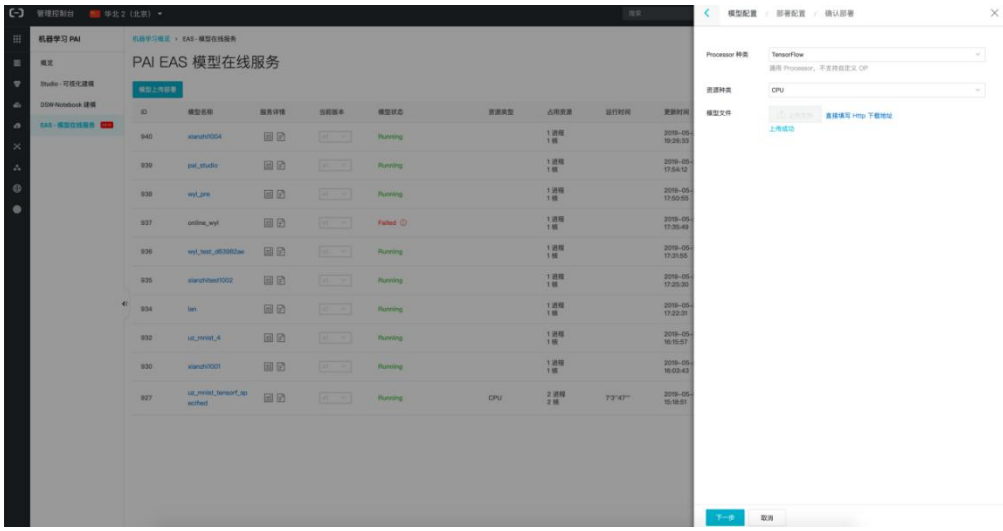
从这个图里也可以看到我们的这个Model里有2个输入源分别叫做images和keep_prob。并且点击它们之后我们还能看到对应的数据格式应该是什么样的。不过没有办法使用 Tensorboard 的同学也不用担心因为EAS这个产品也为我们提供了构造请求的方式。这一次部署我们先使用WEB界面来部署我们的服务这一步也可以通过EASCMD来实现之后我会再写一篇如何用好EASCMD的文章。
我们可以把模型文件下载完之后用zip打包然后到PAI产品的控制台点击EAS-模型在线服务.
ZIP打包可以用这个命令如果你是Unix的用户的话

```
zip -r model.zip path/to/model_files
```

进入EAS之后我们点击模型部署上传



然后继续配置我们的processor这一次因为我们是用tensorflow训练的所以选择Tensorflow然后资源选择CPU有需要的同学可以考虑GPU然后上传我们的模型文件.



点击下一步我们选新建服务然后给我们的服务起个名字,并且配置资源数量。

然后最后确认一下就可以点击部署了

第四步: 调试我们的模型

回到EAS的控制台我们可以看到我们的服务正在被构建中。等到状态显示Running的时候我们就可以开始调试了。

我们可以先点击在线调试。

会让我们跳转到一个Debug 接口的页面。什么都不需要填直接点击提交我们就可以看到服务的数据格式了

然后我们用一段python2的代码来调试这个刚刚部署完的服务。python3的SDK暂时还在研发中。注意要把下面的app_key, app_secret, url 换成我们刚刚部署好的内容。点击模型名字就可以看见了。

其中测试图片的数据大家可以在这下载到。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import json

from urlparse import urlparse
from com.aliyun.api.gateway.sdk import client
from com.aliyun.api.gateway.sdk.http import request
from com.aliyun.api.gateway.sdk.common import constant
from pai_tf_predict_proto import tf_predict_pb2

import cv2
import numpy as np

with open('9.jpg', 'rb') as infile:
    buf = infile.read()
    # 使用numpy将字节流转换成array
    x = np.fromstring(buf, dtype='uint8')
    # 将读取到的array进行图片解码获得28 × 28的矩阵
    img = cv2.imdecode(x, cv2.IMREAD_UNCHANGED)
    # 由于预测服务API需要长度为784的一维向量将矩阵reshape成784
    img = np.reshape(img, 784)

def predict(url, app_key, app_secret, request_data):
    cli = client.DefaultClient(app_key=app_key, app_secret=app_secret)
    body = request_data
    url_ele = urlparse(url)
    host = 'http://' + url_ele.hostname
    path = url_ele.path
    req_post = request.Request(host=host, protocol=constant.HTTP, url=path, method="POST",
time_out=6000)
    req_post.set_body(body)
    req_post.set_content_type(constant.CONTENT_TYPE_STREAM)
    stat,header, content = cli.execute(req_post)
    return stat, dict(header) if header is not None else {}, content


def demo():
    # 输入模型信息,点击模型名字就可以获取到了
    app_key = 'YOUR_APP_KEY'
    app_secret = 'YOUR_APP_SECRET'
    url = 'YOUR_APP_URL'

    # 构造服务
    request = tf_predict_pb2.PredictRequest()
```

```
    request.signature_name = 'predict_images'
    request.inputs['images'].dtype = tf_predict_pb2.DT_FLOAT  # images 参数类型
    request.inputs['images'].array_shape.dim.extend([1, 784])  # images参数的形状
    request.inputs['images'].float_val.extend(img)  # 数据

    request.inputs['keep_prob'].dtype = tf_predict_pb2.DT_FLOAT  # keep_prob 参数的类型
    request.inputs['keep_prob'].float_val.extend([0.75])  # 默认填写一个

    # å°†pbåºåˆ—åŒ–æˆstringè¿›è¡Œä¼ è¾“
    request_data = request.SerializeToString()

    stat, header, content = predict(url, app_key, app_secret, request_data)
    if stat != 200:
        print 'Http status code: ', stat
        print 'Error msg in header: ', header['x-ca-error-message'] if 'x-ca-error-message' in
header else ''
        print 'Error msg in body: ', content
    else:
        response = tf_predict_pb2.PredictResponse()
        response.ParseFromString(content)
        print(response)


if __name__ == '__main__':
    demo()
```

运行这个python代码然后我们会得到

```
outputs {
  key: "scores"
  value {
    dtype: DT_FLOAT
    array_shape {
      dim: 1
      dim: 10
    }
    float_val: 0.0
    float_val: 0.0
    float_val: 0.0
    float_val: 0.0
    float_val: 0.0
    float_val: 0.0
    float_val: 0.0
    float_val: 0.0
    float_val: 0.0
    float_val: 0.0
    float_val: 1.0
  }
}
```

我们可以看到从0开始数的最后一个也就是第9个的结果是1 其他都是0 说明我们的结果是9和我们输入的一样。这样我们就简单轻松的构建了一个在线服务能够将用户的图片中手写数字识别出来。配合其他Web框架或者更多的东西我们就可以作出更好玩的玩具啦。

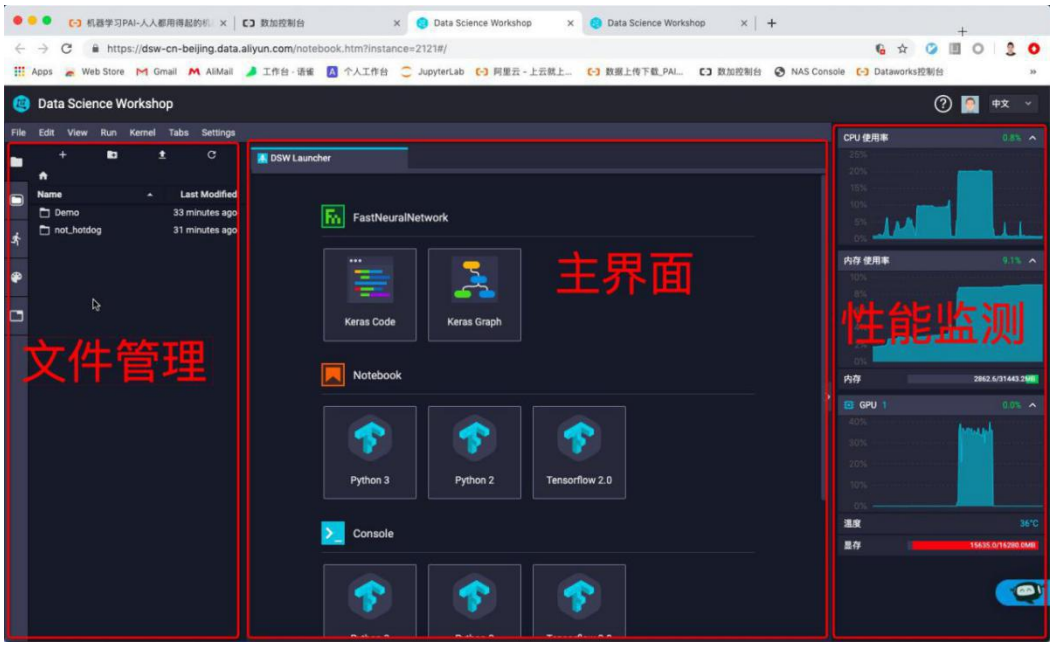
案例三：如何自己训练一个热狗识别模型

美剧《硅谷》大家想必都没怎么看过，大家可能都不知道人工智能识别热狗曾是硅谷最赚钱的技术之一。HBO 曾发布了官方的 Not Hotdog 应用，支持 iOS 和 Android 平台，据说是用 TensorFlow、Keras 和 React Native 打造的，但是源码没有公开。

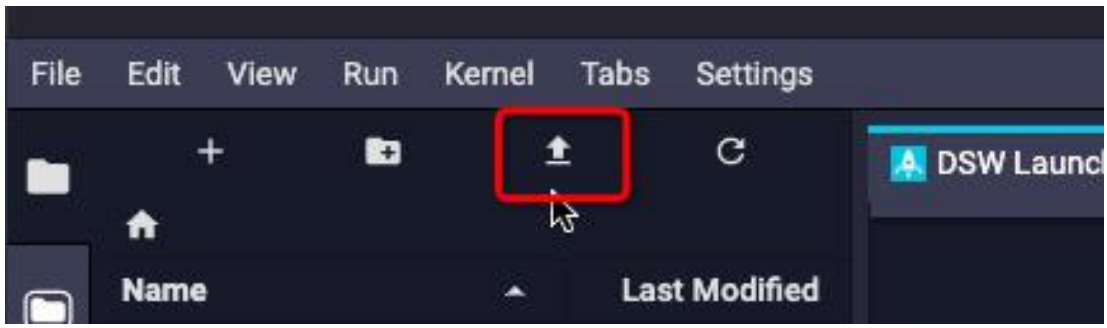
我们今天要做的就是这部美剧里面第四季里面让杨建成为百万富翁的模型：热狗识别模型。这一次，就让阿里云的数据科学老司机带你一起，利用机器学习pai平台训练自己的热狗识别模型，打破技术封锁。让你出任CEO,迎娶白富美/高富帅,走上人生巅峰。

工欲善其事，必先利其器。没有好的工具就想要训练出好的模型简直是天方夜谭。

大家进入DSW环境之后，就可以上传训练代码以及数据了。



我们先从这里下载我为各位精心准备好的代码和训练数据集压缩包。下载到本地之后，点击上传这个按钮 就可以把你的文件上传上来了。



上传成功后，我们打开Terminal 进入这个路径，然后输入

```
$ unzip ./not_hotdog.zip # 解压整个文件
$ cd not_hotdog.zip
$ unzip seefood.zip # 解压训练数据集
```

然后就会看到我们的文件夹已经乖乖躺在我们的左侧的资源管理器里边儿了。

接下来就是我们的硬核部分，我们直接把代码放上来。我们直接运行就可以啦

```
#!/usr/bin/env python
# coding: utf-8

# # Import dependencies 导入依赖

# In[1]:


import numpy as np
import pandas as pd
import os

import tensorflow as tf
rand_state = 42 # 顺便定义一个随机种子
tf.set_random_seed(rand_state)
np.random.seed(rand_state)

from skimage import exposure
import cv2
import glob
import time
import matplotlib.pyplot as plt
from keras.utils.vis_utils import plot_model


# # 图像预处理的函数们

# In[2]:


def rotateImage(img, angle):
  '''
    img:三通道的图片
    angle:随机角度

    本功能是样本增强功能，对图片样本进行随机的旋转缩放

    return:返回一个变换后的图片

    '''

    (rows, cols, ch) = img.shape  # 得到源图片尺寸

    #第一个参数旋转中心，第二个参数旋转角度，第三个参数：缩放比例
```

```
    M = cv2.getRotationMatrix2D((cols/2,rows/2), angle, 1)

    return cv2.warpAffine(img, M, (cols,rows))  # 图像进行上面操作后生成的图像


def loadBlurImg(path, imgSize):
    '''
    path：图片路径，字符串
    imgsize:图片的尺寸,二元组，元素都是int
    '''
    img = cv2.imread(path)  # 读取图片数据
    angle = np.random.randint(0, 360)  # 生成0，360之间生成随机数，离散均匀随机，整形
    img = rotateImage(img, angle)  # 图片随机旋转，缩放
    img = cv2.blur(img,(5,5))     # 每5*5的尺寸进行均值模糊
    img = cv2.resize(img, imgSize)  # 图片按照尺寸缩放
    return img


def loadImgClass(classPath, classLable, classSize, imgSize):
    '''
    classPath:传入图片的路径，list集合
    classLable:图片的类别，数值int
    classSize:样本数量
    imgsize:图片的尺寸,二元组，元素都是int

    return:返回classSize个样本及标签

    本函数从样本地址中生成classSize个数据，样本是经过旋转，缩放等变换的，图片规格是
imgsize

    '''
    x = []
    y = []

    for path in classPath:
        img = loadBlurImg(path, imgSize)  # 加载地址中的图片并进行样本增强，生成imgsize大
的图片
        x.append(img)
        y.append(classLable)

    while len(x) < classSize:

        randIdx = np.random.randint(0, len(classPath))
        img = loadBlurImg(classPath[randIdx], imgSize)
        x.append(img)
        y.append(classLable)

    return x, y

def loadData(img_size, classSize, hotdogs, notHotdogs):
    '''
    img_size:要返回图片的大小,int
    classSize:正例，负例样本数量,int
```

```
    hotsdogs,notHotdogs:正例，负例样本地址，都是个list


    return：返回训练样本及对应的标签

    本函数读取数据并返回样本及标签
    '''

    imgSize = (img_size, img_size)    # 要输入图片的尺寸
    xHotdog, yHotdog = loadImgClass(hotdogs, 0, classSize, imgSize)  # 生成正样本，classSize
个
    xNotHotdog, yNotHotdog = loadImgClass(notHotdogs, 1, classSize, imgSize)  # 生成负样本，
classSize个
    print("There are", len(xHotdog), "hotdog images")
    print("There are", len(xNotHotdog), "not hotdog images")

    X = np.array(xHotdog + xNotHotdog)
    y = np.array(yHotdog + yNotHotdog)

    return X, y

def toGray(images):

    '''
    样本灰度转换，生成后的图片是一个通道的
    '''
    # rgb2gray converts RGB values to grayscale values by forming a weighted sum of the R, G,
and B components:
    # 0.2989 * R + 0.5870 * G + 0.1140 * B
    # source: https://www.mathworks.com/help/matlab/ref/rgb2gray.html

    images = 0.2989*images[:,:,:,0] + 0.5870*images[:,:,:,1] + 0.1140*images[:,:,:,2]
    return images

def normalizeImages(images):
    '''
    images:1个通道的图像
    return：图像像素经过比例缩放，直方图均衡后的图像
    '''
    # use Histogram equalization to get a better range
    # source http://scikit-
    image.org/docs/dev/api/skimage.exposure.html#skimage.exposure.equalize_hist
    images = (images / 255.).astype(np.float32)  # rgb像素是0-255之间，缩放至0-1的范围

    for i in range(images.shape[0]):
        images[i] = exposure.equalize_hist(images[i])  # 直方图均衡之后的图像数组

    images = images.reshape(images.shape + (1,))  #  二维扩成三维
    return images

def preprocessData(images):
```

```
    '''
    images:三通道的image
    return:返回一通道，且数值经过比例缩放的图片（除以255，使之数值范围集中在0-1之
间）
    '''
    grayImages = toGray(images)
    return normalizeImages(grayImages)


# # 我们需要对图像做一些骚操作 毕竟500张图片还是太少了

# In[3]:


from keras.utils.np_utils import to_categorical
from sklearn.model_selection import train_test_split

size = 32
classSize = 20000

# In[7]:


# 导入数据
hotdogs = glob.glob('./train/hot_dog/**/*.jpg', recursive=True)
notHotdogs = glob.glob('./train/not_hot_dog/**/*.jpg', recursive=True)


# In[12]:


dd = (20000,20000)
print(dd)


# In[14]:


# 骚操作一波
scaled_X, y = loadData(size, classSize, hotdogs, notHotdogs)
scaled_X = preprocessData(scaled_X)


# In[15]:


y = to_categorical(y)   # 目标变量独热


n_classes=2
print("y shape", y.shape)
X_train, X_test, y_train, y_test = train_test_split(
    scaled_X,
    y,
    test_size=0.2,
```

```
 random_state=rand_state
)   # 数据按照训练集0.8的比例分割


print("train shape X", X_train.shape)
print("train shape y", y_train.shape)
print("Test shape X:", X_test.shape)
print("Test shape y: ", y_test.shape)

inputShape = (size, size, 1)


# In[8]:


def plot_history(history):
    loss_list = [s for s in history.history.keys() if 'loss' in s and 'val' not in s]
    val_loss_list = [s for s in history.history.keys() if 'loss' in s and 'val' in s]
    acc_list = [s for s in history.history.keys() if 'acc' in s and 'val' not in s]
    val_acc_list = [s for s in history.history.keys() if 'acc' in s and 'val' in s]

    if len(loss_list) == 0:
        print('Loss is missing in history')
        return

    ## As loss always exists
    epochs = range(1,len(history.history[loss_list[0]]) + 1)

    ## Loss
    plt.figure(1)
    for l in loss_list:
        plt.plot(epochs, history.history[l], 'b', label='Training loss (' +
str(str(format(history.history[l][-1],'.5f'))+')'))
    for l in val_loss_list:
        plt.plot(epochs, history.history[l], 'g', label='Validation loss (' +
str(str(format(history.history[l][-1],'.5f'))+')'))

    plt.title('Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    ## Accuracy
    plt.figure(2)
    for l in acc_list:

        plt.plot(epochs, history.history[l], 'b', label='Training accuracy (' +
str(format(history.history[l][-1],'.5f'))+')')
    for l in val_acc_list:
        plt.plot(epochs, history.history[l], 'g', label='Validation accuracy (' +
str(format(history.history[l][-1],'.5f'))+')')

    plt.title('Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
```

```
  plt.show()


# # 重点来了：构建模型就是这儿了


# In[9]:


import keras
from keras.models import Sequential
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.layers import Conv2D, MaxPooling2D, Dense, Dropout, Flatten
from keras.layers.normalization import BatchNormalization


model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
            activation='relu',
            kernel_initializer='he_normal',
            input_shape=inputShape))  # 卷积
model.add(MaxPooling2D((2, 2)))          # 池化
model.add(Dropout(0.25))                 # 随机失活
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(Dropout(0.4))
model.add(Flatten())                     # 展成一维
model.add(Dense(128, activation='relu'))  # 全连接
model.add(Dropout(0.3))
model.add(Dense(2, activation='softmax'))

model.compile(loss=keras.losses.binary_crossentropy,
          optimizer=keras.optimizers.Adam(lr=1e-4),
          metrics=['accuracy'])

start = time.time()

model.summary()
# Set callback functions to early stop training and save the best model so far
callbacks = [
  EarlyStopping(
    monitor='val_loss',
    patience=3

  ),
  ModelCheckpoint(
    filepath='model.h5',
    monitor='val_acc',
    save_best_only=True
  )
]
```

```
history = model.fit(
    X_train,
    y_train,
    batch_size=32,
    epochs=100,
    callbacks=callbacks,
    verbose=0,

    validation_data=(X_test, y_test)
)

end = time.time()
print('Execution time: ', end-start)

plot_history(history)
```

训练完成之后，我们可以简单的测试一下我们模型的准确率。下面这段代码就可以帮我们做到这一点。

```
hotdogs = glob.glob('./test/hot_dog/**/*.jpg', recursive=True)
notHotdogs = glob.glob('./test/not_hot_dog/**/*.jpg', recursive=True)

scaled_X_test, y_test = loadData(size, 250, hotdogs, notHotdogs)
scaled_X_test = preprocessData(scaled_X_test)

#get the predictions for the test data
predicted_classes = model.predict_classes(scaled_X_test)

# setup the true classes: just 250 hotdogs followed by 250 not hotdogs
y_true = np.concatenate((np.zeros((250,)), np.ones((250,))))
from sklearn.metrics import classification_report
print(classification_report(y_true, predicted_classes, target_names=['hotdog', 'not hotdog']))
```

这样我们就可以看到我们模型的比较重要的一些评估结果了，比如准确率什么的。

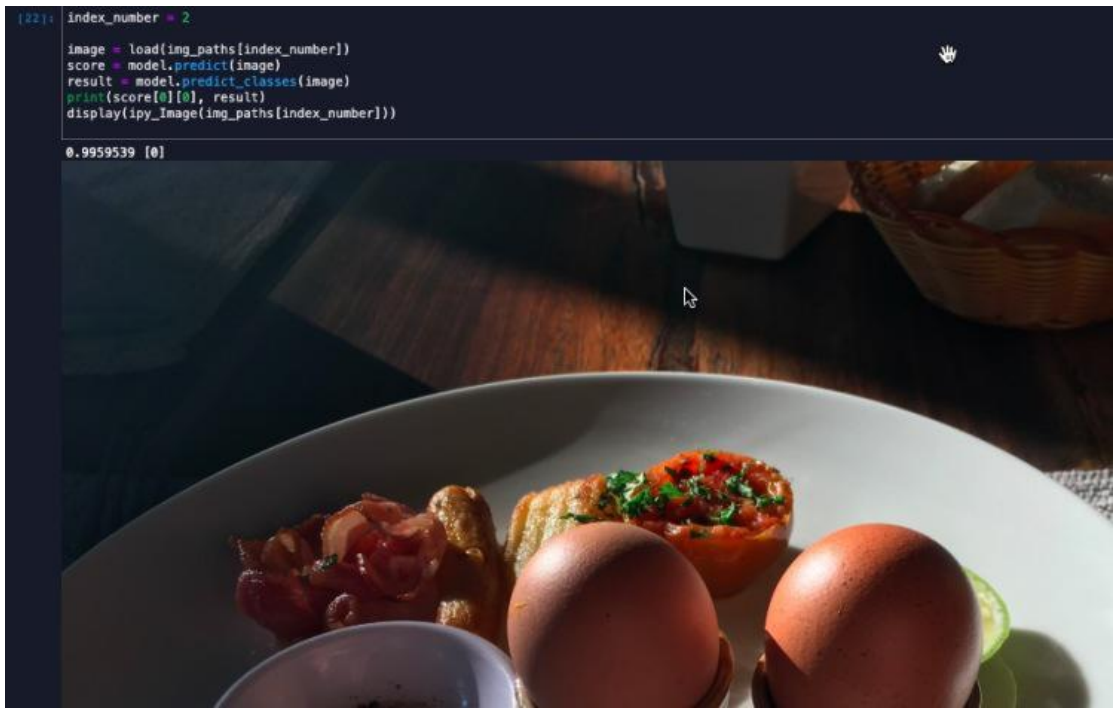
但是我们既然辛辛苦苦训练了，我们就要好好把玩一下这个模型。我们可以直接用下面这段代码来预测一个图片里面是不是有热狗。在这之前需要我们先创建一个名叫**foo**的文件夹，并把你想要测试的图片放进去

```
from PIL import Image
import numpy as np
from skimage import transform


from IPython.display import Image as ipy_Image
from IPython.display import display

# 定义一个加载图片的函数，使我们的图片变成np array
def load(filename):
  np_image = Image.open(filename)
  np_image = np.array(np_image).astype('float32')/255
```

```
  np_image = transform.resize(np_image, (32, 32, 1))
  np_image = np.expand_dims(np_image, axis=0)
  return np_image

import os
from os.path import join

image_dir = './foo'

os.listdir(image_dir)
img_paths = [join(image_dir,filename) for filename in os.listdir(image_dir)]

index_number = 0

image = load(img_paths[index_number])
score = model.predict(image)
result = model.predict_classes(image)
print(score[0][0], result)
display(ipy_Image(img_paths[index_number]))
```

比如我们这里上传一张直播中网友们发来的图片，这张图片在直播的时候成功骗过了模型，得分最高

我们运行一下，就可以看到结果了



我们可以看到这个图片完美骗过了我们的模型，几乎达到了1。大家也可以拿这个模型测试一下自己身边长的像是热狗但是又不是热狗的东西，看看到底能得多少分～

案例四：半小时验证语音降噪—贾扬清邀你体验快捷云上开发

语音降噪，在开源领域通过科学计算肯定有现有的解决方案。从业务需求上讲，最有价值的一件事情是，怎么让大家能迅速地做POC，架起解决方案？

本文将实战讲解贾扬清在回答内部同学提出的业务问题时，给自己设的一个挑战，半个小时架构一个有体感的demo，达到语音降噪的效果。

半小时实验达到的语音降噪的效果

首先给大家看一下最终的效果。我录了一个关着吸尘器的时候的语音：

In [2]:

```
import IPython
IPython.display.Audio('https://notebook-dataset.oss-cn-beijing.aliyuncs.com/noises/normal.wav')
```

Out[2]:

https://developer.aliyun.com/topic/download?id=826

以及开着吸尘器的时候讲话的语音：

In [3]:

```
IPython.display.Audio('https://notebook-dataset.oss-cn-beijing.aliyuncs.com/noises/voice_with_noise.wav')
```

Out[3]:

https://developer.aliyun.com/topic/download?id=827

以及，通过降噪以后，带着吸尘器的那一段音频的降噪效果：

In [4]:

```
IPython.display.Audio('https://notebook-dataset.oss-cn-beijing.aliyuncs.com/noises/voice_reduced_noise.wav')
```

Out[4]:

https://developer.aliyun.com/topic/download?id=828

效果怎么样？

在实际业务当中，我们的思路往往不是上来就开始钻研算法，而是按照这样的一个思路：

- 怎么将业务问题翻译成技术问题？
- 有没有现有解决方案来测试一下效果？
- 效果好的话，怎么落地？
- 效果不好的话，怎么做算法迭代？

今天想展示的就是通过DSW快速解决前两个问题，即如何迅速安装开源的算法包、做数据的可视化、到最后算法效果的测试这整个流程。用现有的平台迅速地验证这些业务问题的效果，是不是一件很有意思的事情呢？

通过DSW快速验证

首先，通过搜索，我们发现有一个github的项目，noisereduce，和我们需要解决的场景很相似。基本上，可以通过两段语音，一段是噪音（做建模），一段是带噪音的语音，来实现降噪的效果。

对于标准的Python算法包，在DSW里面拉起很容易。因为DSW预装了底层的一些科学计算框架，比如说TensorFlow，因此只需要安装新增的这些包的需求：

In [5]:

```
!pip install noisereduce > /dev/null
```

我拿iPhone和家里的吸尘器录了几段视频，上传到DSW上面，然后就是大家常见的码代码了：

In [6]:

```
from matplotlib import pyplot
import io
import noisereduce as nr
import numpy as np
import soundfile as sf
from urllib.request import urlopen
```

```
/Users/huanghong/opt/anaconda3/lib/python3.7/site-packages/tqdm/autonotebook.py:17: TqdmExperimentalWarning: Using `tqdm.autonotebook.tqdm` in notebook mode. Use `tqdm.tqdm` instead to force console mode (e.g. in jupyter console) " (e.g. in jupyter console)", TqdmExperimentalWarning)
```

In [7]:

```
# 先把音频文件导入进来：
normal_data, normal_rate = sf.read(io.BytesIO(urlopen('https://notebook-dataset.oss-cn-beijing.aliyuncs.com/noises/normal.wav').read()))
noise_data, noise_rate = sf.read(io.BytesIO(urlopen('https://notebook-dataset.oss-cn-beijing.aliyuncs.com/noises/noise.wav').read()))
mixed_data, mixed_rate = sf.read(io.BytesIO(urlopen('https://notebook-dataset.oss-cn-beijing.aliyuncs.com/noises/voice_with_noise.wav').read()))
# iPhone的音频是立体声的，我们只处理一个声道，所以我们先选单声道。
normal_data = np.ascontiguousarray(normal_data[:,0])
noise_data = np.ascontiguousarray(noise_data[:,0])
mixed_data = np.ascontiguousarray(mixed_data[:,0])
rate = normal_rate
```

我们可以用Notebook的功能来播放这些音频。在开头大家已经听过样例了，这里我们听一下噪声：

In [8]:

```
IPython.display.Audio(data=noise_data,rate=rate)
```

Out[8]:

https://developer.aliyun.com/topic/download?id=829

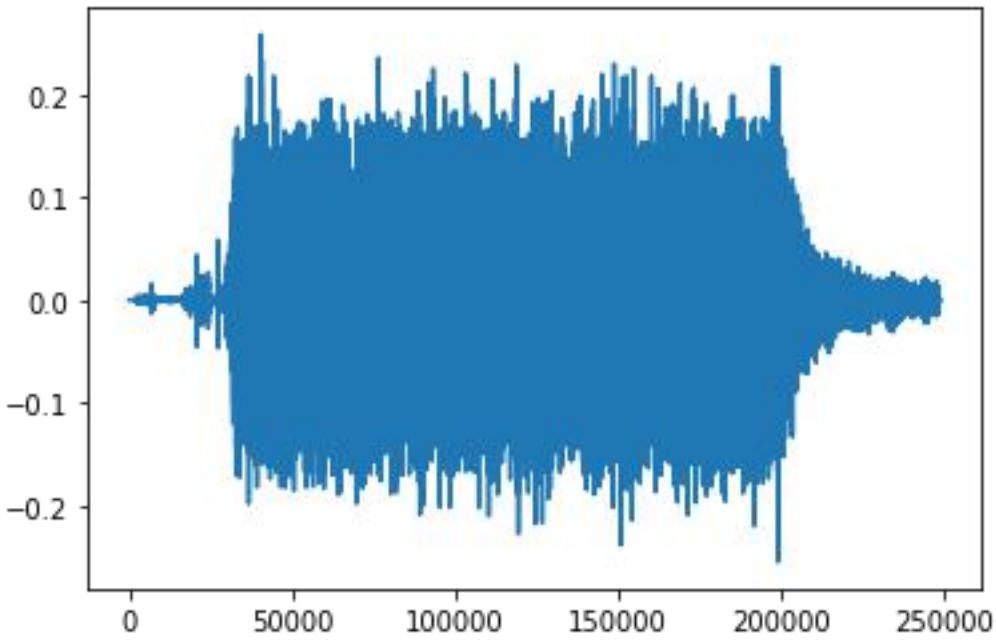
对于噪声文件，我们来截取中间一段作为噪声的建模。先看看，哪段比较合适。

In [9]:

```
pyplot.plot(noise_data)
```

Out[9]:

[<matplotlib.lines.Line2D at 0x1c29049c10>]



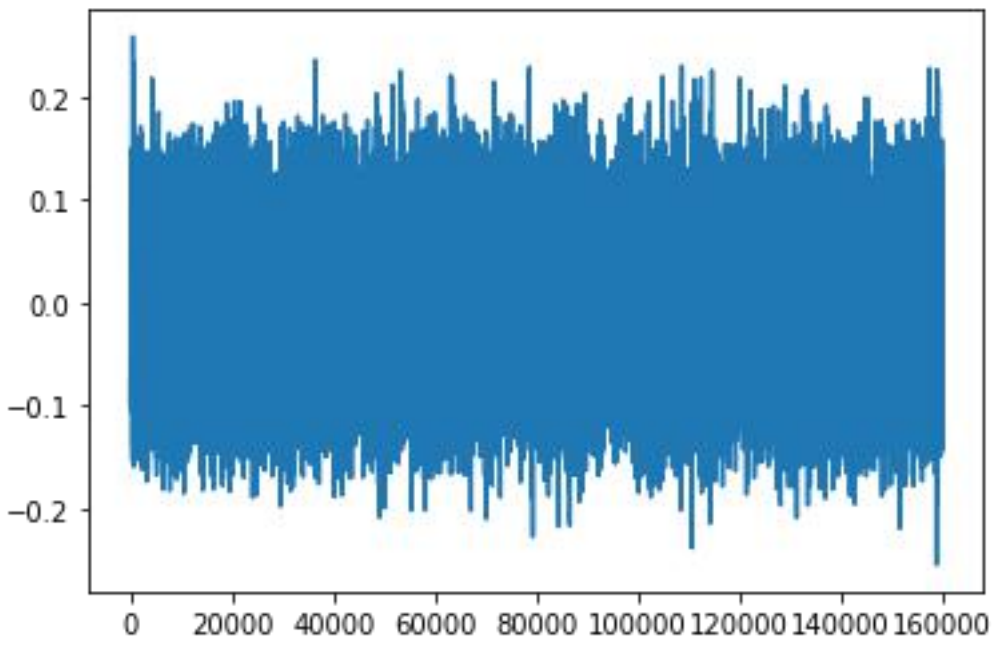
看来，截取40000到200000的这一段最合适，我们剪一下。

In [10]:

```
noise_data = noise_data[40000:200000]
pyplot.plot(noise_data)
```

Out[10]:

[<matplotlib.lines.Line2D at 0x1c290b5190>]



然后，我们就可以直接调用一下noisereduce里面的功能看看结果怎么样。一行代码：

In [11]:
```
reduced_version = nr.reduce_noise(audio_clip=mixed_data,
noise_clip=noise_data)
```

我们来听一下，降噪之后的语音效果怎么样。

In [12]:
```
IPython.display.Audio(data=reduced_version,rate=rate)
```

Out[12]:
https://developer.aliyun.com/topic/download?id=830

再听一下降噪之前的版本。

In [13]:
```
IPython.display.Audio(data=mixed_data,rate=rate)
```

Out[13]:
https://developer.aliyun.com/topic/download?id=831

我们可以把降噪之后的版本存下来：

In [14]:
```
sf.write('voice_reduced_noise.wav', reduced_version, rate)
```

算法是AI的核心，但是要实现业务，功夫在算法之外。

大家可以看到，在上面的样例当中，其实核心算法很短：首先，有大量的开原算法可以让我们快速验证现有的技术在不同场景当中的价值；其次，很多应用并不一定在第一时间就需要特别高大上的算法。对于关注业务的工程师来说，如何快速做POC，验证可行性，然后从浅到深来把算法落地，做算法创新，实现业务落地，这是我们今天关注的重点。

今天，一切业务都会数据化，一切数据都会业务化。相应的，一切应用就会数据化和智能化。我们对AI的看法可以用英语来做一下类比：几十年前，英语是一种服务，我们雇佣专业的翻译来帮我们做业务；但是今天，英语是一种工具，我们绝大多数人用各种工具来学习英语，使用英语。

这就对平台提出了更高的要求。**PAI一直向“最懂你的AI平台”不断努力。**DSW（Data Science Workshop）这个产品，目标是给大家一个云上托管的，易用开放的机器学习和深度学习开发平台，让大家很容易地能够拉起一个典型的开发环境，迅速投入到算法和应用的开发当中。