**3.1 特征聚合**

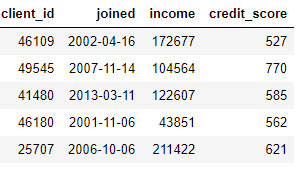
机器学习越来越多地从人工设计模型转向使用 H20、TPOT 和 auto-sklearn 等工具自动优化的工具。这些库以及随机搜索（参见《Random Search for Hyper-Parameter Optimization》）等方法旨在通过寻找匹配数据集的最优模型来简化模型选择和机器学习调优过程，而几乎不需要任何人工干预。然而，特征工程作为机器学习流程中可能最有价值的一个方面，几乎完全是人工的。

特征工程也被称为特征构造，是从现有数据中构造新的特征从而训练机器学习模型的过程。这一步可能比实际上使用的模型更重要，因为一个机器学习算法只能从我们给定的数据中学习，所以构造一个和任务相关的特征是至关重要的，参见优质论文《A Few Useful Things to Know about Machine Learning》。通常，特征工程是一个冗长的人工过程，依赖于领域知识、直觉和数据操作。这个过程可能是极其枯燥的，同时最终得到的特征将会受到人的主观性和时间的限制。特征工程自动化旨在通过从数据集中自动构造候选特征，并从中选择最优特征用于训练来帮助数据科学家。

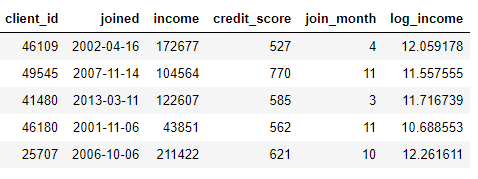
在本文中，我们将介绍一个使用 Feature Tools Python 库实现特征工程自动化的例子。我们将使用一个示例数据集来说明基本概念（继续关注之后使用真实世界数据的例子）。

**3.1.1 特征工程基本概念**

特征工程意味着从现有的数据中构造额外特征，这些特征通常分布在多张相关的表中。特征工程需要从数据中提取相关信息并将其存入单张表格中，然后被用来训练机器学习模型。构造特征是一个非常耗时的过程，因为每个新的特征通常需要几步才能构造，特别是当使用多张表的信息时。我们可以将特征构造的操作分为两类：「转换」和「聚合」。以下通过几个例子来看看这些概念的实际应用。通过从一或多列中构造新的特征，「转换」作用于单张表（在 Python 中，表是一个 Pandas DataFrame）。举个例子，若有如下的某一个县城客户表：



我们可以通过查找 joined 列中的月份或是自然对数化 income 列的数据来构造新的特征。这些都是转换操作，因为它们只用到了一张表的信息。



另一方面，「聚合」是跨表实现的，并使用一对多的关联来对观测值分组，然后计算统计量。例如，若我们有另外一张包含另一个县城客户贷款信息的表格，其中每个客户可能有多项贷款，我们便可以计算每个客户贷款的平均值、最大值和最小值等统计量。这个过程包括根据不同客户对贷款表进行分组并计算聚合后的统计量，然后将结果整合到客户数据中，构成整个市级单位的表格。以下是我们在 Python 中使用 Pandas 库执行此操作。

import pandas as pd

# Group loans by client id and calculate mean, max, min of loans

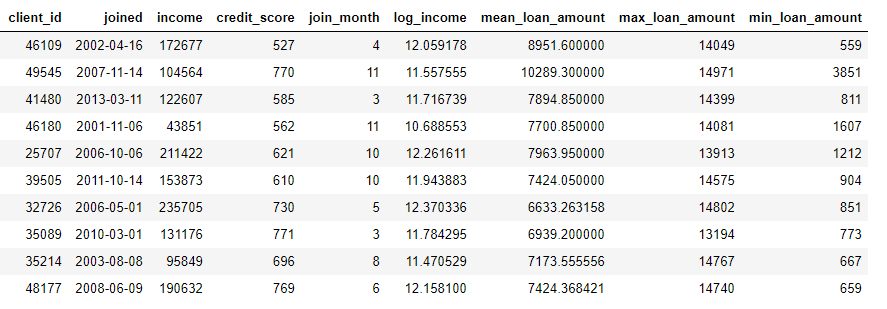
stats = loans.groupby('client\_id')['loan\_amount'].agg(['mean', 'max', 'min'])

stats.columns = ['mean\_loan\_amount', 'max\_loan\_amount', 'min\_loan\_amount']

# Merge with the clients dataframe

stats = clients.merge(stats, left\_on = 'client\_id', right\_index=True, how = 'left')

stats.head(10)



这些操作本身并不困难，但是如果有数百个变量分布在数十张表中，这个过程将无法通过人工完成。理想情况下，我们希望有一个解决方案能够在不同表间自动执行转换和聚合操作，并将结果整合到一张表中。尽管 Pandas 是一个很好的资源，但是仍然有许多数据操作需要我们人工完成！有关人工特征工程的更多信息，请查阅《Python Data Science Handbook》。

**3.1.2特征聚合的目的**

目的：

* 数据简化：通过将表格数据合并，减少特征数。
* 变换尺度：低级数据向高级数据转化，例如，将县级银行贷款数据转化为市级银行贷款。
* 更加稳定的数据：使数据有更少的变化，例如将月份数据转化为年数据。

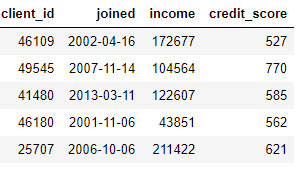
**3.1.3特征工具**

幸运的是，Feature Tools 正是我们正在找寻的解决方案。这个开源的 Python 库可以从一组相关的表中自动构造特征。特征工具基于名为「深度特征合成」的方法（参见《Deep Feature Synthesis: Towards Automating Data Science Endeavors》），这个方法的名字听起来比其本身更高大上（这个名字源于叠加了多重特征，而不是因为使用了深度学习方法！）。

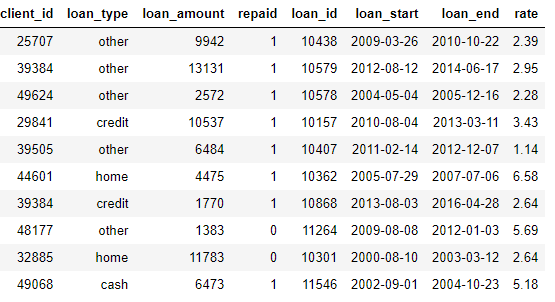
深度特征合成叠加多个转换和聚合操作，这在特征工具的词库中被称为特征基元，以便通过分布在多张表内的数据来构造新的特征。与机器学习中的大多数方法一样，这是建立在简单概念基础之上的复杂方法。通过一次学习一个构造块，我们可以很好地理解这个强大的方法。

首先，让我们看一下示例数据。我们已经看到了上面的一些数据集，并且完整的表组如下所示：

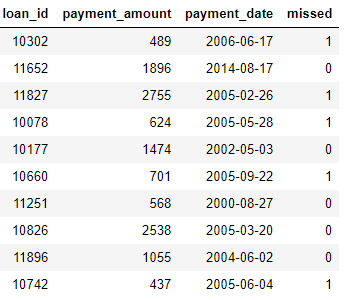
* clients： 关于信用社客户的基本信息。每个客户只对应数据框中的一行。



* loans： 向用户提供的贷款。每项贷款只对应数据框中的一行，但是客户可能有多项贷款。



* payments：贷款还本的支付。每笔支付只对应一行，但是每项贷款可以有多笔支付。



如果我们有一个机器学习任务，例如预测客户未来是否会偿还一项贷款，我们希望将所有关于客户的信息整合到一张表中。这些表是相关的（通过 client\_id 和 loan\_id 变量），并且我们可以通过一系列转换和聚合操作来人工实现这个过程。然而，我们很快就可以使用特征工具来自动实现这个过程。

**实体和实体集**

特征工具的前两个概念的是「实体」和「实体集」。一个实体就是一张表（或是 Pandas 中的一个 DataFrame（数据框））。一个实体集是一组表以及它们之间的关联。将一个实体集看成另一种 Python 数据结构，并带有自己的方法和属性。

我们可以通过以下操作在特征工具中创建一个空的实体集：

import featuretools as ft

# Create new entityset

es = ft.EntitySet(id = 'clients')

现在我们需要整合两个实体。每个实体都必须带有一个索引，它是一个包含所有唯一元素的列。就是说，索引中的每个值只能在表中出现一次。在 clients 数据框中的索引是 client\_id，因为每个客户在该数据框中只对应一行。我们使用以下语法将一个带有索引的实体添加一个实体集中：

# Create an entity from the client dataframe

# This dataframe already has an index and a time index

es = es.entity\_from\_dataframe(entity\_id = 'clients', dataframe = clients,

                              index = 'client\_id', time\_index = 'joined')

loans 数据框还有另外一个唯一的索引，loan\_id，同时将其添加到实体集的语法与 clients 一样。然而，payments 数据框不存在唯一索引。当我们把 payments 数据框添加到实体集中时，我们需要传入参数make\_index = True，同时指定索引的名字。另外，尽管特征工具能自动推断实体中每列的数据类型，但是我们可以通过将列数据类型的字典传递给参数 variable\_types 来覆盖它。

# Create an entity from the payments dataframe

# This does not yet have a unique index

es = es.entity\_from\_dataframe(entity\_id = 'payments',

                              dataframe = payments,

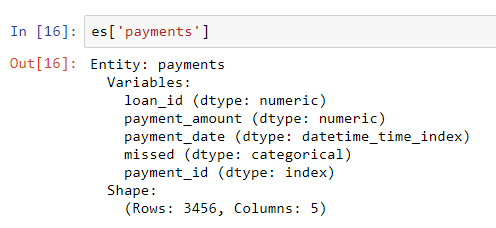
                              variable\_types = {'missed': ft.variable\_types.Categorical},

                              make\_index = True,

                              index = 'payment\_id',

                              time\_index = 'payment\_date')

对于此数据框，尽管 missed 是一个整数，但是它不是一个数值变量，因为它只能取 2 个离散的数值，所以在特征工具中，将其看成一个分类变量。在将该数据框添加到实体集中后，我们检查整个实体集：



列的数据类型已根据我们指定的修正方案被正确推断出来。接下来，我们需要指定实体集中表是如何关联的。

**表的关联**

考虑两张表之间「关联」的最好方法是类比父子之间的关联。这是一种一对多的关联：每个父亲可以有多个儿子。对表来说，每个父亲对应一张父表中的一行，但是子表中可能有多行对应于同一张父表中的多个儿子。

例如，在我们的数据集中，clients 数据框是 loans 数据框的一张父表。每个客户只对应 clients 表中的一行，但是可能对应 loans 表中的多行。同样，loans 表是 payments 表的一张父表，因为每项贷款可以有多项支付。父亲通过共享变量与儿子相关联。当我们执行聚合操作的时候，我们根据父变量对子表进行分组，并计算每个父亲的儿子的统计量。

为了形式化特征工具中的关联规则，我们仅需指定连接两张表的变量。clients 表和 loans 表通过 client\_id 变量连接，同时 loans 表和 payments 表通过 loan\_id 变量连接。创建关联并将其添加到实体集中的语法如下所示：

# Relationship between clients and previous loans

r\_client\_previous = ft.Relationship(es['clients']['client\_id'],

                                    es['loans']['client\_id'])

# Add the relationship to the entity set

es = es.add\_relationship(r\_client\_previous)

# Relationship between previous loans and previous payments

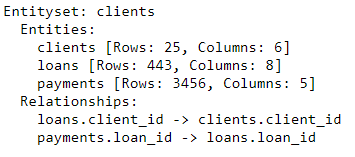
r\_payments = ft.Relationship(es['loans']['loan\_id'],

                                      es['payments']['loan\_id'])

# Add the relationship to the entity set

es = es.add\_relationship(r\_payments)

es



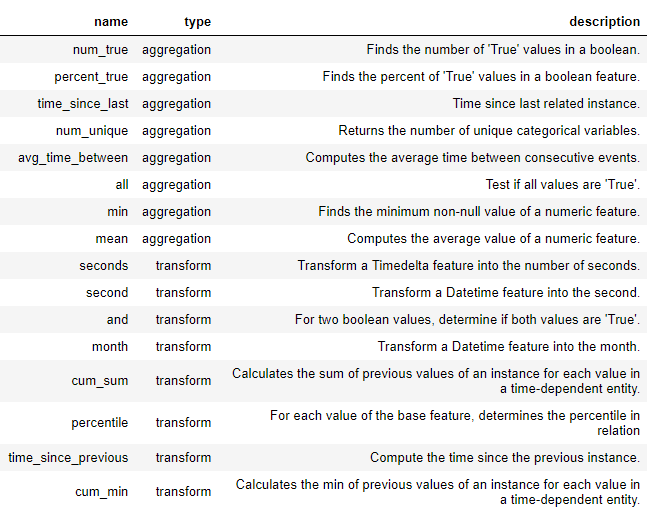
该实体集现在包含三个实体（表），以及将这些表连接在一起的关联规则。在添加实体和形式化关联规则之后，实体集就完整了并准备好从中构造新的特征。

**特征基元**

在我们深入了解深度特征合成之前，我们需要了解特征基元的概念。我们其实早就知道是什么了，只是我们刚刚用不同的名字来称呼它们！它们只是我们用来构造新特征的操作：

* 聚合：根据父与子（一对多）的关联完成的操作，也就是根据父亲分组并计算儿子的统计量。一个例子就是根据 client\_id 对 loan 表分组并找到每个客户的最大贷款额。
* 转换：对一张表中一或多列完成的操作。一个例子就是取一张表中两列之间的差值或者取一列的绝对值。

在特征工具中单独使用这些基元或者叠加使用这些基元可以构造新的特征。以下是特征工具中一些特征基元的列表，也可以自定义特征基元。



这些基元可以单独使用或是组合使用以构造新的特征。为了使用特定的基元构造新的特征，我们使用 ft.dfs 函数（代表深度特征合成）。我们传入 entityset 和 target\_entity，这是我们想要在其中添加特征的表，被选参数 trans\_primitives（转换）和 agg\_primitives（聚合）。

public class MyActivity extends AppCompatActivity {

@Override  //override the function

    protected void onCreate(@Nullable Bundle savedInstanceState) {

       super.onCreate(savedInstanceState);

       try {

            OkhttpManager.getInstance().setTrustrCertificates(getAssets().open("mycer.cer");

            OkHttpClient mOkhttpClient= OkhttpManager.getInstance().build();

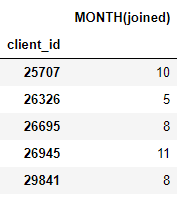
        } catch (IOException e) {

            e.printStackTrace();

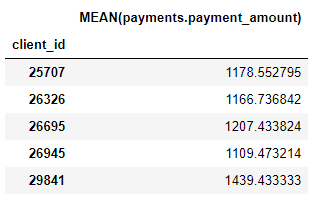
        }

}

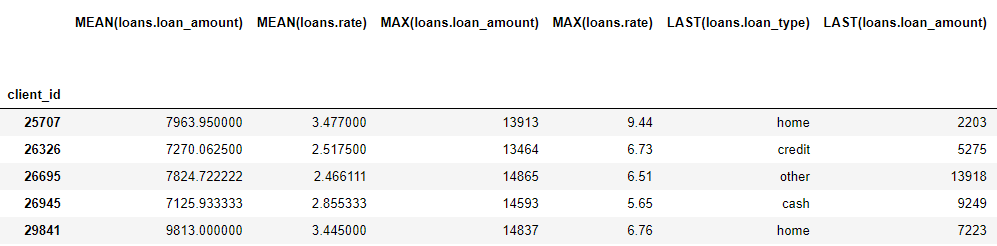
返回的是包含每个客户新特征的数据框（因为我们定义客户为 target\_entity）。比如我们有每个客户加入的月份，这是一个转换操作的特征基元：



我们也有许多聚合操作的基元，比如每个客户的平均支付总额：



尽管我们仅指定了一些特征基元，但是特征工具可以通过组合和叠加这些基元来构造新的特征。

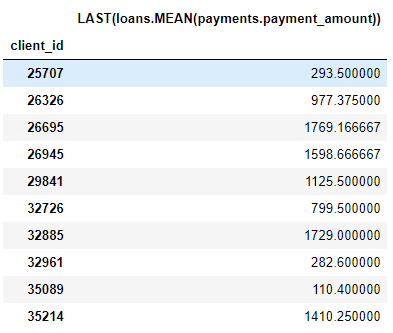


完整的数据框包含 793 列的新特征！

**深度特征合成**

我们现在具备理解深度特征合成（dfs）的一切条件。事实上，我们已经在前面的函数调用中执行了 dfs！深度特征只是叠加多个基元构造的一个特征，而 dfs 只是构造这些特征的过程的名称。深度特征的深度是构造这个特征所需的基元数量。

例如，MEAN（payments.payment\_amount）列是深度为 1 的特征，因为它是使用单个聚合操作构造的。LAST（loans（MEAN（payments.payment\_amount））是一个深度为 2 的特征，它是由两个叠加的聚合操作构造的：MEAN 列之上的 LAST（最近的）列。这表示每个客户最近的贷款平均支付额。



我们可以叠加任意深度的特征，但在实践中，我从没有使用超过 2 个深度的特征。此外，这些特征很难解释，但是我鼓励任何对「深入」感兴趣的人。

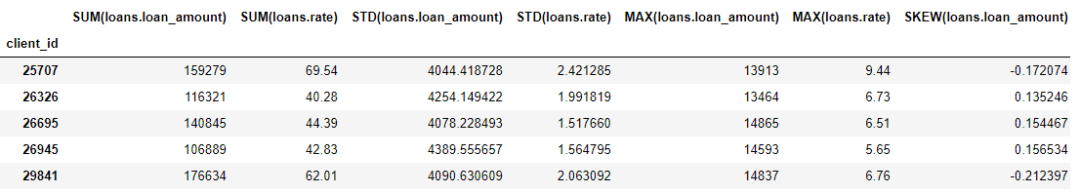
我们不必人工指定特征基元，但可以让特征工具自动为我们选取特征。为此，我们使用相同的 ft.dfs 函数调用，但是不传入任何特征基元。

# Perform deep feature synthesis without specifying primitives

features, feature\_names = ft.dfs(entityset=es, target\_entity='clients',

                                 max\_depth = 2)

features.head()



特征工具构造了很多特征供我们使用。尽管这个过程确实能自动构造新的特征，但是它不会取代数据科学家，因为我们仍然需要弄清楚如何处理这些特征。例如，我们的目的是预测一位客户是否会偿还贷款，我们可以寻找与特定结果最相关的特征。此外，如果我们具有领域知识，我们可以用这些知识来选择指定的特征基元或候选特征的种子深度特征合成。

特征工程自动化解决了一个问题，但是带来了另一个问题：特征太多了。尽管在拟合一个模型之前很难说哪些特征是重要的，但很可能不是所有这些特征都与我们想要训练的模型的任务相关。此外，拥有太多特征（参见《Irrelevant Features and the Subset Selection Problem》）可能会导致模型性能不佳，因为较无益的特征会淹没那些更重要的特征。

特征过多问题以维度灾难著称。随着特征数量的上升（数据维度增长），模型越来越难以学习特征与目标之间的映射关系。事实上，让模型表现良好所需的数据量与特征数量成指数关系。

维度灾难与特征降维（也叫特征选择，去除不相关特征的过程）相对。这可以采用多种形式：主成分分析（PCA）、SelectKBest、使用模型中特征的重要性或使用深度神经网络进行自编码。但是，特征降维是另一篇文章的不同主题。到目前为止，我们知道我们可以使用特征工具以最小的努力从许多表中构造大量的特征！

**3.1.4结论**

与机器学习中的许多主题一样，使用特征工具进行特征工程自动化是一个基于简单想法的复杂概念。使用实体集、实体和关联的概念，特征工具可以执行深度特征合成操作来构造新的特征。深度特征合成可以依次叠加特征基元：「聚合」，它们在多张表间的一对多关联中起作用，以及「转换」，是应用于单张表中一或多列以从多张表中构造新的特征的函数。