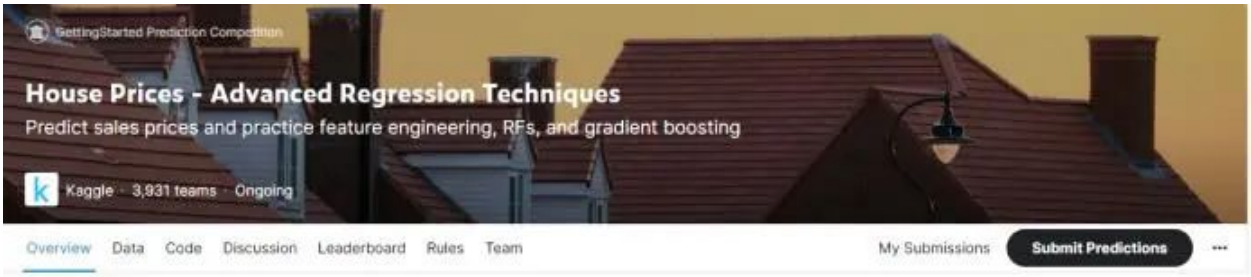
如何在Kaggle入门赛中拿到铜牌（bushi

原创 做饭饭 三文鱼卷 2022-09-19 17:30 发表于北京

收录于合集
#学习指南

14个



写在前面

在阅读之前你需要知道的是，本文包含了大量**机器学习**和**深度学习**领域专业词汇，适合有一定基础的同学用来了解kaggle，对于初学者来说并不友好。

首先说说比赛，这是kaggle上给新手练手的一个入门级比赛，有近4000队伍参加。比赛提供给你csv格式的train data和test data，以此来建立一个模型预测结果，并在网站上提交。在**房价预测**这个比赛中，你会得到一些有关房子的数据，如建造日期、住宅风格等，以此来预测其最终的售价。由于是入门赛，每天都能交10次，教程也很多，所以各路神仙齐聚甚至把分刷到了0.0000（说明预测的结果和真实值一丁点都没有差）。

那么言归正传，让我们看看各位大佬是如何得到一个很高的分数的。我分别看了一下机器学习的方法（最终结果0.12左右，251/3949），也调了一下神经网络(结果0.13左右, 1000/3949)。发现在这种回归预测的比赛里集成学习的表现非常优秀，深度学习炼丹还是有一些困难的。

01神经网络

首先试了一下神经网络，模型基于**pytorch**，主体代码部分参考了**李沐老师**的。（是跟李沐学ai，而不是19级的那位....）代码就不放在正文里了，单独看很难看懂，了解了思路再看代码也不迟。

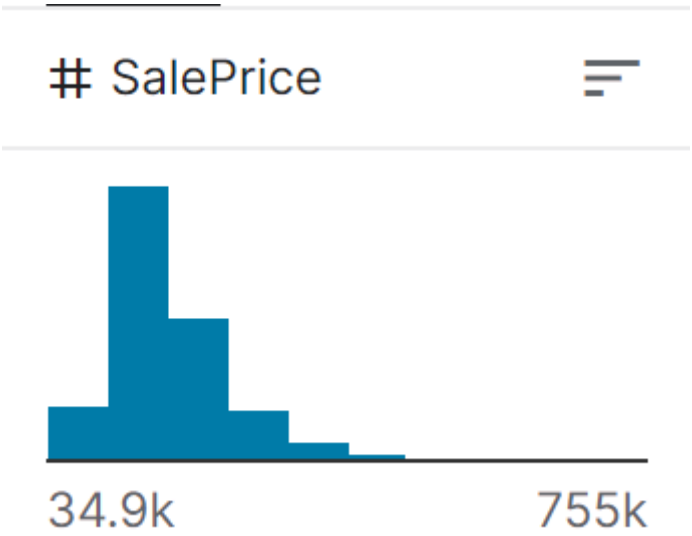
关于读取数据和数据泄露

首先是简单的读取数据和划分验证集，我使用了20%所给的train data作为验证集，其余的为训练集，由于直接取了前百分之二十而不是random sample，所以可能会有一些过拟合。（但由于懒得改就没管了）

在李沐老师的代码中，他用了**k-fold**来对模型的效果进行验证，其每折的验证集都是从预处理之后的train data中抽取的。这就会导致所谓的数据泄露(data leakage), 也即在你的验证过程中，你的训练集数据对验证数据造成了影响。具体的原因是比如在填补train data的NA值时用mean进行填补。如果不提前**对训练集和验证集进行划分**，那么出现在你验证数据中的缺失值会变成训练集和验证集整体的平均值，而不是验证集中单独的平均值。这无疑是降低了模型的泛用性。同时创建新特征或者特征编码也会导致**data leakage**。

虽然好像data leakage听起来很严重，但是事实上很多选手不仅没有提前划分验证集和训练集，甚至他们把比赛中给的train data和test data合并到一起进行插补和特征编码等操作。这种虽然很大程度导致了模型对于现有数据的**过拟合**，一旦遇到将来的数据可能啥也不是。但架不住真有用啊，非要弄个泛用性好的在比赛的榜单上也体现不出来。所以这个data leakage在比赛中可能不需要过度考虑。

价格向量的处理



价格向量SalePrice是需要我们预测的东西。观察其在train data的分布，可以发现其明显地不是正态分布，同时最大值和最小值差距很大。如果不对其处理的话可能预测结果并不理想，所以在神经网络中，我对其使用了zscore进行处理。但是预测最终结果的时候还是需要训练集的mean和std进行，所以可能导致最后在训练集表现不错，一到测试集就变拉。

神经网络结构

```
class Net(nn.Module):
    global num_inputs, num_outputs, num_hiddens1, num_hiddens2, dropout1, dropout2
    '''
    the net framework
    '''
    def __init__(self, num_inputs, num_outputs, num_hiddens1, num_hiddens2, dropout1, dropout2):
        super(Net, self).__init__()
        self.num_inputs = num_inputs
        self.lin1 = nn.Linear(num_inputs, num_hiddens1)
        self.lin2 = nn.Linear(num_hiddens1, num_hiddens2)
        self.lin3 = nn.Linear(num_hiddens2, num_outputs)
        self.relu = nn.ReLU()
        self.dropout1 = dropout1
        self.dropout2 = dropout2


    def forward(self, X):
        X = self.relu(self.lin1(X.reshape((-1, self.num_inputs))))
        # 在全连接层之后添加一个dropout层
        X = F.dropout(X, self.dropout1)
        X = self.relu(self.lin2(X))
        X = F.dropout(X, self.dropout2)
        X = self.lin3(X)
        return X
```

由于还不太会用*args和**kwargs，并且写这点代码也没必要，所以直接用了**全局变量**，到时候调参也比较好改。网络只有一层隐藏层，虽然大伙儿说两层比较好，但当时数错了，一直以为有两层隐藏层就没加。激活函数和优化也是经典的**ReLU**和**Adam**组合，为什么这个组合好我也说不清，反正是李沐老师说的。

其中还加了两层**dropout**，虽然最后没有起作用。因为我发现在把price取zscore进行计算后，如果再启用dropout，哪怕是0.1也会导致学习曲线产生很剧烈的波折。我的大略猜测是取zscore导致结果都在很小的范围内，而神经元内的参数都比较大，使得一旦一些神经元不激活就会给结果造成很大程度的影响。

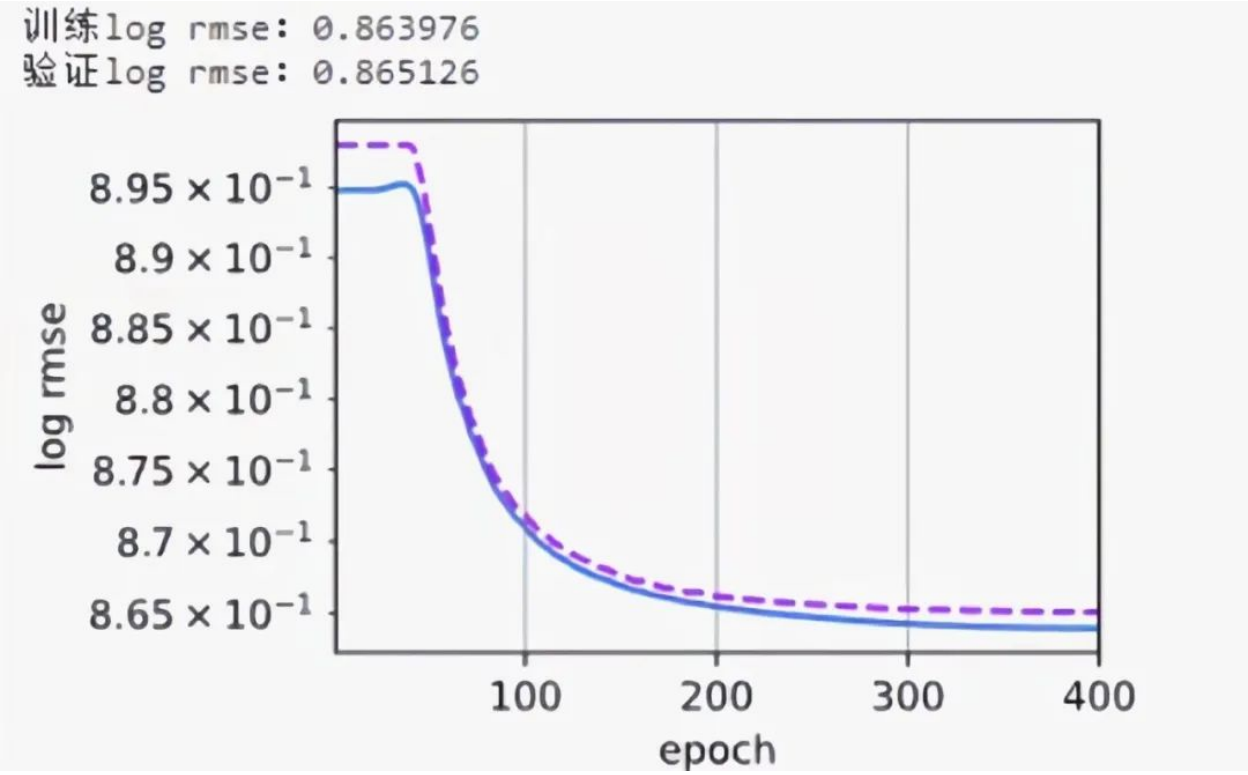
Loss和learning rate等参数设置

```
loss = nn.SmoothL1Loss()
batch_size = 128
k = 5

def learning_rate(temp_epochs):
    global epochs,start_decline_epochs
    if temp_epochs < start_decline_epochs: return 5e-6
    else:
        return 5e-6*math.cos((temp_epochs-start_decline_epochs)/(epochs-start_decline_epochs)*math.pi/2)

def learning_rate(temp_epochs):
    global epochs,start_decline_epochs
    if temp_epochs < start_decline_epochs: return 1e-5
    else:
        return 1e-5-1e-5 *(temp_epochs)/epochs
```

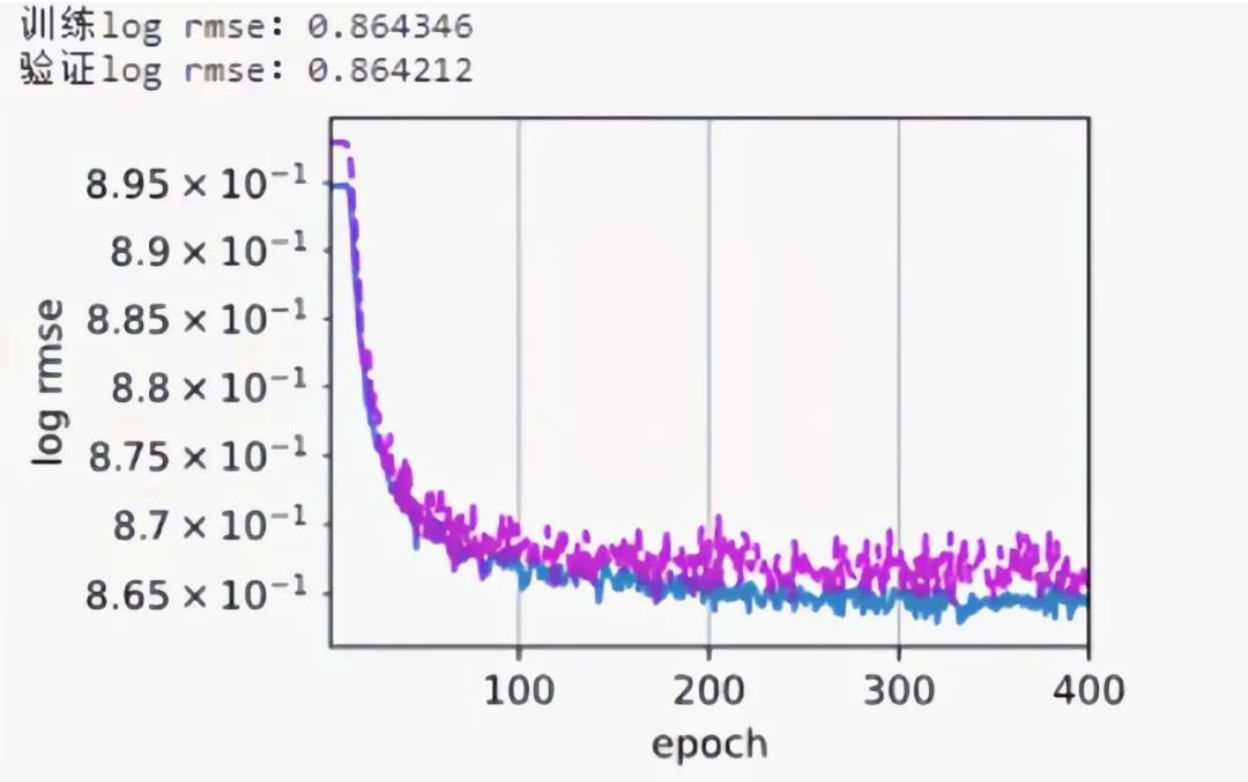
Loss是 SmoothL1Loss，没有用MSE的原因是因为MSE名字不够长（bushi）。是因为看到某篇文章说这个函数更加体面，实际上我简单换了一下没有很大的区别。

Batch size 是128。Batch size 越大训练速度越快，但是变小后训练集的准确率会提高一些。我又调了一下调成32和16，训练的log rmse到了0.861左右。

Learning rate写了个函数用来及时更改学习率的变化。据李沐老师所说，学习率像cos一样减少可以使学习率变化的更加平滑，效果更好。但是我试下来感觉没啥用，~~可能因为是我自创的学习率函数不体面~~。



下图为启用dropout



最后结果

最后的学习曲线我觉得还挺好看的，虽然结果不是很理想，但是再调也很费劲了。不过听高人指点，调出好的学习曲线会有更好的结果。

02机器学习

相较于神经网络而言，机器学习对于特征工程更为重视，可以说是机器学习的最关键的点就是特征工程和集成学习了。虽然有人说有的比赛给的样本数量比较少，用神经网络也不能将所有有用的特征组合起来，所以也需要进行一定的特征工程。我大概试了一下，不能说是效果明显，只能说 log rmse简直下降了一个数量级，所以深度学习中可能也要重视一下**特征工程**。

特征工程

对于特征原作者做了很多工作，虽然很多是借鉴别人的，还有很多操作莫名其妙，并且在他的文章中也缺乏注释。但是给我也是带来了很大的启发。首先是一种巧妙的编码方法：

```
def encode(frame, feature):
    ordering = pd.DataFrame()
    ordering['val'] = frame[feature].unique()
    ordering.index = ordering.val
    ordering['spmean'] = frame[[feature, 'SalePrice']].groupby(feature).mean()['SalePrice']
    ordering = ordering.sort_values('spmean')
    ordering['ordering'] = range(1, ordering.shape[0]+1)
    ordering = ordering['ordering'].to_dict()

    for cat, o in ordering.items():
        frame.loc[frame[feature] == cat, feature+'_E'] = o

qual_encoded = []
for q in qualitative:
    encode(train, q)
    qual_encoded.append(q+'_E')
print(qual_encoded)
```

这段代码的大致含义就是将所有的**离散型特征**比如房屋的风格之类的，按不同变量将样本进行分组计算均值，并进行排序，按照这个顺序再创建一个变量。然而这里虽然我觉得是个好点子，但是之后的处理就太抽象了。在之后的过程中，作者把test data和加了变量后的train data硬生生的concat在一起，然后把test data中缺失的这些encode feature都填上了0.0.我觉得这个可能是~~狗尾续貂，瞎猫碰死耗子~~，当然也不排除作者故意给了个错误代码坑人。

```
features['MSSubClass'] = features['MSSubClass'].apply(str)
features['YrSold'] = features['YrSold'].astype(str)
features['MoSold'] = features['MoSold'].astype(str)
features['Functional'] = features['Functional'].fillna('Typ')
features['Electrical'] = features['Electrical'].fillna("SBrkr")
features['KitchenQual'] = features['KitchenQual'].fillna("TA")
features["PoolQC"] = features["PoolQC"].fillna("None")
features['Exterior1st'] = features['Exterior1st'].fillna(features['Exterior1st'].mode()[0])
features['Exterior2nd'] = features['Exterior2nd'].fillna(features['Exterior2nd'].mode()[0])
features['SaleType'] = features['SaleType'].fillna(features['SaleType'].mode()[0])
```

这里是先把一些数字的离散变量变成**字符串**，比如卖出的年份和月份之类的，以免在构建模型时过于依赖这些特征（我猜的。之后的一些填充大部分应该都是用众数、中位数或者0进行填充，这些单拎出来我也不知道是何用意，可能是别人代码抄着抄着就懒得改了。

```
numeric_dtypes = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
numerics2 = []
for i in features.columns:
    if features[i].dtype in numeric_dtypes:
        numerics2.append(i)
skew_features = features[numerics2].apply(lambda x: skew(x)).sort_values(ascending=False)

high_skew = skew_features[skew_features > 0.5]
skew_index = high_skew.index

for i in skew_index:
    features[i] = boxcox1p(features[i], boxcox_normmax(features[i] + 1))
```

之后对于一些**高偏度**（high skewness）的特征，作者又对其应用box cox使其变为正态分布。（我：数据泄露不考虑吗？作者：我不到啊！）

```
features = features.drop(['Utilities', 'Street', 'PoolQC',], axis=1)

features['YrBltAndRemod']=features['YearBuilt']+features['YearRemodAdd']
features['TotalSF']=features['TotalBsmtSF'] + features['1stFlrSF'] + features['2ndFlrSF']

features['Total_sqr_footage'] = (features['BsmtFinSF1'] + features['BsmtFinSF2'] +
                                 features['1stFlrSF'] + features['2ndFlrSF'])

features['Total_Bathrooms'] = (features['FullBath'] + (0.5 * features['HalfBath']) +
                               features['BsmtFullBath'] + (0.5 * features['BsmtHalfBath']))

features['Total_porch_sf'] = (features['OpenPorchSF'] + features['3SsnPorch'] +
                              features['EnclosedPorch'] + features['ScreenPorch'] +
                              features['WoodDeckSF'])
```

接下来作者莫名其妙的删除了三个特征，明明PoolQC这个特征的spearman相关系数还算高，也没有注释之类的。这里通过多个特征的加和构建了新的特征，这些特征大多是对于预测有现实意义的。比如是卫生间总面积，就是一堆与卫生间面积有关的特征的加和。说起来这个步骤是不是放到encode之前做会好一些，这样又多出了5个新特征。

```
features['haspool'] = features['PoolArea'].apply(lambda x: 1 if x > 0 else 0)
features['has2ndfloor'] = features['2ndFlrSF'].apply(lambda x: 1 if x > 0 else 0)
features['hasgarage'] = features['GarageArea'].apply(lambda x: 1 if x > 0 else 0)
features['hasbsmt'] = features['TotalBsmtSF'].apply(lambda x: 1 if x > 0 else 0)
features['hasfireplace'] = features['Fireplaces'].apply(lambda x: 1 if x > 0 else 0)
```

接着又构建了这些是否有什么什么的逻辑值特征，~~看上去非常的体面~~，我都不知道说什么能填满这一行了。之后作者简单的get_dummies获得了离散特征的one-hot编码。

```
overfit = []
for i in X.columns:
    counts = X[i].value_counts()
    zeros = counts.iloc[0]
    if zeros / len(X) * 100 > 99.94:
        overfit.append(i)

overfit = list(overfit)
X = X.drop(overfit, axis=1)
X_sub = X_sub.drop(overfit, axis=1)
```

这里的**overfit**不是模型过拟合的意思，而是查看如果某个特征如果他99.94%以上都是一种特征，那我们就把他删掉。

对于标签作者也做了处理，应用np.log1p将其化为近似**正态分布**。然后预测值再用np.expm1就可以将其转化为实际的价格了。至此特征工程结束，开始模型构建。

模型构建

我其实用了**h2o autoML**尝试了一下，发现过拟合太严重了。严重到我都怀疑是不是我代码写错了，但在反复验证调试多次后我发现可能并不是我的问题。我现在复盘时觉得应该是那个大愚若智的编码让模型在训练集表现良好，结果一到测试集全是0就给模型整不会了。（后来经过测试发现确实是这样的，改了一下编码后再用h2o效果非常体面）

```
def blend_models_predict(X):
    return ((0.1 * elastic_model_full_data.predict(X)) + \
            (0.05 * lasso_model_full_data.predict(X)) + \
            (0.1 * ridge_model_full_data.predict(X)) + \
            (0.1 * svr_model_full_data.predict(X)) + \
            (0.1 * gbr_model_full_data.predict(X)) + \
            (0.15 * xgb_model_full_data.predict(X)) + \
            (0.1 * lgb_model_full_data.predict(X)) + \
            (0.3 * stack_gen_model.predict(np.array(X))))
```

作者模型大概就是用**GridSearch**或者别的什么参数搜索方法挨个训练出最好的各类不同模型，并混合到一起，以防过拟合。（过拟合到不同方向等于不过拟合，是这样的）

```
sub_1 = pd.read_csv('../input/top-10-0-10943-stacking-mice-and-brutal-force/House_Prices_submit.csv')
sub_2 = pd.read_csv('../input/hybrid-svm-benchmark-approach-0-11180-lb-top-2/hybrid_solution.csv')
sub_3 = pd.read_csv('../input/lasso-model-for-regression-problem/lasso_sol22_Median.csv')
submission.iloc[:,1] = np.floor((0.25 * np.floor(np.expm1(blend_models_predict(X_sub)))) +
                                (0.25 * sub_1.iloc[:,1]) +
                                (0.25 * sub_2.iloc[:,1]) +
                                (0.25 * sub_3.iloc[:,1]))
```

这是最让我觉得离谱的部分了。如果我的理解没错的话他把别人的预测结果加进来和自己的结果混合了一下。这就让我想到了那句，~~我和马云加起来富可敌国~~...虽然作者本身模型做的也不错。

```
q1 = submission['SalePrice'].quantile(0.005)
q2 = submission['SalePrice'].quantile(0.995)
submission['SalePrice'] = submission['SalePrice'].apply(lambda x: x if x > q1 else x*0.77)
submission['SalePrice'] = submission['SalePrice'].apply(lambda x: x if x < q2 else x*1.1)
submission.to_csv("submission.csv", index=False)
```

在提交前他还整了个活儿，在**top**和**bottom**的少部分样本可能是离群的不体面的点，他就把特别大的变小，特别小的变大，损有余而补不足了属于是。

结语

Kaggle真是个有趣的平台，当你发现排名上涨时会非常有成就感。同时，在此致谢**许燕老师**解答了这篇文章中一些问题。路漫漫其修远兮，吾将上下而求索。

文案|做饭饭

排版|半月

收录于合集 #学习指南 14

下一篇 · 自学雅思的n种方法

喜欢此内容的人还喜欢

学习指南 | 大一上两航类培养方案保姆级教学

三文鱼卷