Kaggle Bike Sharing项目报告

标签（空格分隔）： ML

项目预览

项目地址

> https://www.kaggle.com/c/bike-sharing-demand

项目简介:

> Bike sharing systems are a means of renting bicycles where the process of obtaining membership, rental, and bike return is automated via a network of kiosk locations throughout a city. Using these systems, people are able rent a bike from a one location and return it to a different place on an as-needed basis. Currently, there are over 500 bike-sharing programs around the world.
>
> The data generated by these systems makes them attractive for researchers because the duration of travel, departure location, arrival location, and time elapsed is explicitly recorded. Bike sharing systems therefore function as a sensor network, which can be used for studying mobility in a city. In this competition, participants are asked to combine historical usage patterns with weather data in order to forecast bike rental demand in the Capital Bikeshare program in Washington, D.C.

要求是通过历史数据来预测（包括时间、天气等因素）某一天的租车数。
选取这个题目的原因是我认为这个问题和机器学习还是紧密相关的，与之前做过的project相似，需要对数据进行处理之后选取模型进行预测。通过解决这个问题可以在一定程度上帮助Bike sharing站点设置等方面还是有一定帮助的。也能让个人判断什么时候租车的人可能会比较多，由此来安排自己的出行。

项目分析

项目提供了三个csv文件，分别是train.csv训练集，test.csv测试集，和sampleSubmission.csv上传文件格式.我们训练用到的是前两个文件。经过前面的分析，我认为这应该是一个监督学习中的回归问题。因为我们要输出一个个的离散值。我们通过数据采用一定的算法构建模型后，对test集中的数据进行预测。由于预测值的区间范围较大，因此采用网站给的评价准则，即：

Evaluation

Submissions are evaluated one the Root Mean Squared Logarithmic Error (RMSLE). The RMSLE is calculated as

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i+1)-\log(a_i+1))^2}$$

Where:

- n is the number of hours in the test set
- p_i is your predicted count
- a_i is the actual count
- $\log(x)$ is the natural logarithm

这个值越小越好。这个评价模型中同时考虑的预测值与实际值的差距，并没有考虑运行时间。为了满足RMSLE的条件，考虑在数据中对数据取对数之后在去e指数可以尽可能的满足此标准。
因此我在本项目中打算采用2~3个算法模型对数据进行拟合，选取精度较高的模型之后在采取修正提高精度。

问题分析

通过调用pd.read_csv()函数，读取train和test数据集。
通过

```
len(train)
len(test)
len(train.columns)
train.head()
```

得到训练集共10886条数据，测试集共6493条数据。共19个feature，分别为

```
[datetime, season, holiday, workingday, weather, temp, atemp, humidity,
windspeed]
```

以及输出的结果

```
[casual, registered, count]
```

其中count为需要预测的值，casual和registered分别为登记和未登记的人数，加起来总和等于count，因此可以将其删除掉。
对特征进行分析，其中日期变量给出的是整体时间，考虑到租车与月份和日期和小时对预测是相对重要的，在接下来的数据处理中需要将其拆分成年/月/日/小时以及周几的形式。
季节特征(season)，在训练集是以category变量给出的，可以考虑将其转变成01变量。对租车的预测应该也是比较重要的。
天气变量(weather)同上。
训练集中给了temp/atemp两个变量，根据对atemp描述可知：

> atemp - “feels like” temperature in Celsius

atemp是一个主观变量，和temp(即温度变量)有很高的相关性，考虑在数据处理中将其移除。
计算count的均值、中位数、四分位数。得到：

```
              count
count  10886.000000
mean     191.574132
std      181.144454
min        1.000000
25%       42.000000
50%      145.000000
75%      284.000000
max      977.000000
```

实施解决方案

1. 数据处理

首先按照上文所述，先对训练集和测试集进行Feature Engineering。
代码如下：

```
train['year'] = train['datetime'].str.extract("^(.{4})")
test['year'] = test['datetime'].str.extract("^(.{4})")

train['month'] = train['datetime'].str.extract("-(.{2})-")
test['month'] = test['datetime'].str.extract("-(.{2})-")

train['day'] = train['datetime'].str.extract("(.{2}) ")
test['day'] = test['datetime'].str.extract("(.{2}) ")

train['time'] = train['datetime'].str.extract(" (.{2})")
test['time'] = test['datetime'].str.extract(" (.{2})")
```

```
train[['year', 'month', 'day', 'time']] = train[['year', 'month', 'day',
'time']].astype(int)
test[['year', 'month', 'day', 'time']] = test[['year', 'month', 'day',
'time']].astype(int)

train['dayOfWeek'] = train.apply(lambda x: datetime.date(x['year'],
x['month'], x['day']).weekday(), axis=1)
test['dayOfWeek'] = test.apply(lambda x: datetime.date(x['year'], x['month'],
x['day']).weekday(), axis=1)

train = train.drop(labels=["datetime"], axis=1)
test = test.drop(labels=["datetime"], axis=1)
```

上述代码作用是处理datetime这一feature，将其转化为我们需要的年月日小时周几，将其转化为int类型后删除datetime。
对weather和season的处理即是对其使用get dummies函数使其生成多个0/1分类feature。

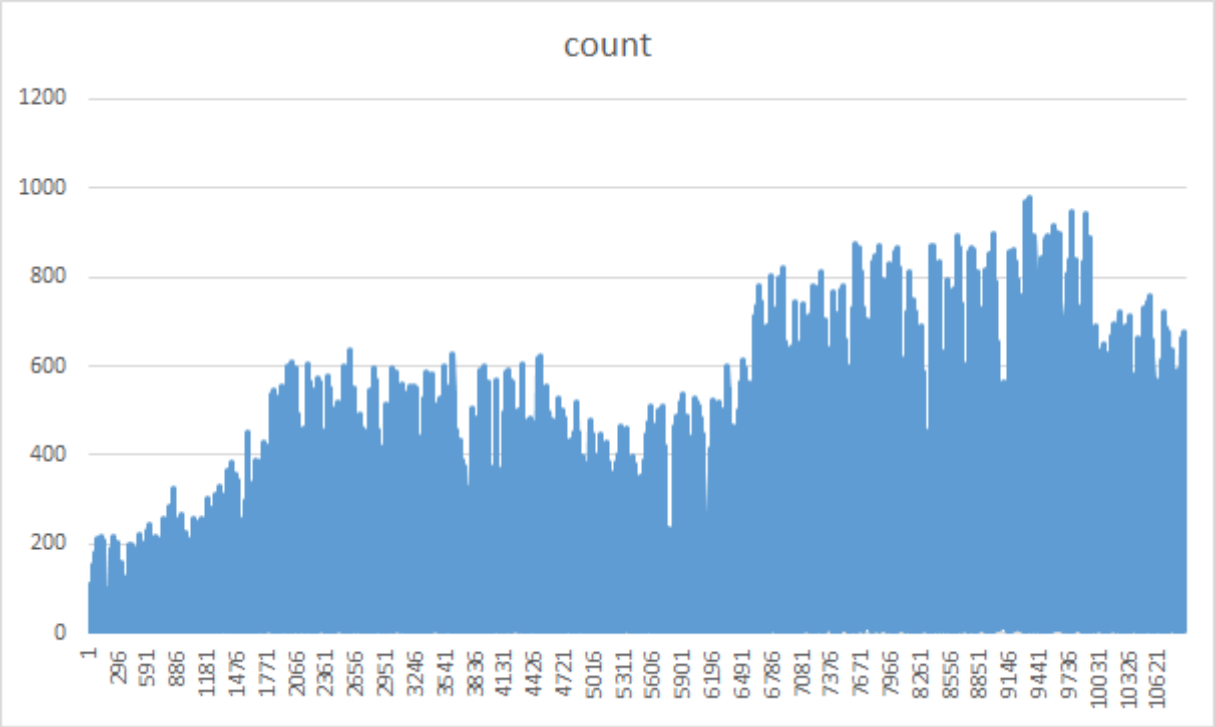
```
train['season'].value_counts()
train = train.join(pd.get_dummies(train.season, prefix='season'))
test = test.join(pd.get_dummies(test.season, prefix='season'))


train = train.drop(labels=["season"], axis=1)
test = test.drop(labels=["season"], axis=1)


train['weather'].value_counts()
train = train.join(pd.get_dummies(train.weather, prefix='weather'))
test = test.join(pd.get_dummies(test.weather, prefix='weather'))


train = train.drop(labels=["weather"], axis=1)
test = test.drop(labels=["weather"], axis=1)
```

数据处理完了之后，考虑到检验准确率的方式采用的是RMSLE，对train数据集取对数，进行预测之后再用指数函数变换回来。
根据数据分布图可知count数据范围变化不大，无须对count数据进行缩放变换。



2.算法选取

考虑到本项目中是使用回归预测模型值，决定采用SVM、决策树，以及将决策树与聚成学习结合的随机森林算法。
SVM我之前知道的大多数用于分类，这里用于回归不知道效果如何。
决策树算法比起线性回归要好得多，但也容易出现过拟合。而且当出现缺失值的情况时容易出现误判。
随机森林算法结合了决策树和继承学习的优点，其计算简单容易实现。但是其最终集成的泛化性能会随着个体差异而增大，不过在本项目中足够使用了。

3.建立模型并预测
为方便算法比较，生成一个函数读入回归模型，train数据，test数据来生成结果。

```
def bs_fit_and_save(clf, l_train, l_target, l_test, filename):

    clf.fit (l_train, l_target)

    predict_train = clf.predict(l_train)

    print('Variance score: %.2f' % clf.score(l_train, l_target))

    predict_test = clf.predict(l_test)
    predict_test = np.exp(predict_test)

    output = test_orig['datetime']
    output = pd.DataFrame(output)
    predict = pd.DataFrame(predict_test)
    output = output.join(predict)
    output.columns = ['datetime', 'count']
    output.to_csv(filename + ".csv", index=False)
    return clf
```

输出score来简单判断模型预测情况。不过此时的score只是显示对train数据的拟合程度。

这里decisionTree采用默认设置
randonForest参数设置n_estimators=100，
SVR设置C=1.0, epsilon=0.2

> decisionTree: Variance score: 1.00
> randomForest: 0.99
> SVR: 0.90
> 对分数分析，发现决策树很可能出现过拟合。随机森林和决策树分数都不错，SVR最低。再看一下kaggle给出的分数。
> 决策树分数

-	danache	0.56008

> 随机森林分数

-	danache	0.43617

> SVM分数：

-	danache	1.34572

> 由于在这里分数越低表现模型误差越小，因此可以看出来SVM最差，并不适合做此类分析，随机森林最好，决策树由于出现了过拟合情况表现也不佳

模型改进

使用交叉验证，先对决策树使用使其max_depth参数范围为[1:10]，得道德结果score为

> Variance score: 0.96

kaggle验证分数为

-	danache	0.51382

相较于之前的0.56，提升了0.5左右，还是很明显的。
由于随机森林无需交叉验证，我用过改变n_estimators参数来修正模型，n_estimators从range(50,151,5)中选取。得到结果

> 50:0.43693
> 55:0.43583
> 60: 0.43599
> 70:0.43596
> 75:0.43655
> 80:0.43646
> 90:0.43621
> 90:0.43645

```
105:0.43528
110:0.43528
120:0.43540
130:0.43459
```

根据数据发现改变forest中tree的数量只能使预测误差微调，如果不改变数据的处理方式的话使用随机森林方法不能在本质上提升模型结果。由于tree的数目对结果影响不是现行的，因此我只能在上述参数中选取最优的。
svm方法由于初始结果太差，就没有对其再进行参数调整。
因此综上采用随机森林的方法可以较好的预测结果，最好的分数为
0.43459。

结果

通过对数据的处理、选择模型、实施验证并且采用交叉验证改变参数后，最终获得的最优结果在kaggle排名698名。通过本项目我意识到数据处理是及机器学习中的重要组成部分，好的数据处理可以是的预测精度大大增加。不同的算法模型预测出来的结果也相差很大。因此仔细斟酌题目类型选取合适的算法是很重要的。