数据预处理

首先使用 `df.info()` 查看数据集的信息。

- 认为 id 一般来说不包含有意义的数据，所以先删除项 `id` 和 `id_str`。而有两个域 `utc_offset` 和 `time_zone` 对所有数据均为 `null`，应该直接删除。

```
d.drop(["id", "id_str", "utc_offset", "time_zone"], axis=1, inplace=True)
```

- `entities` 域包含的数据格式较为复杂，难以解析。另外，有几个包含字符串和 URL 的域，解析它们需要较为复杂的技术，可以先不处理，若之后效果需要提升，再考虑引入这些信息。

在做这些处理之后，所有数据都是 `non-null` 的。考虑处理那些非整形的数据。

- 有两个名字都是 `created_at` 的重名域，但它们是不同的数据，推测第一个是数据的创建时间，没有意义，将其删除。第二个域的形式被判断为 `object`，将其转换为 `datetime` 格式，之后再将 `datetime` 转换为以天为单位的 `float` 数据。

```
d["created_at"] = pd.to_datetime(d["created_at"], infer_datetime_format=True)
d["created_at"] = d["created_at"].apply(lambda x: x.value // (10**9) / (24 * 60 * 60))
```

- 将标签信息单独提取为整形数据。将颜色的 RGB 值分开转换为整形。

```
df_label = pd.get_dummies(df["label"]).iloc[:, 0]
df.drop(["label"], axis=1, inplace=True)
```

```
d_rgb = d[name].apply(col2rgb)
d_rgb.columns = [name+"_r", name+"_g", name+"_b"]
d = pd.concat([d.drop([name], axis=1), d_rgb], axis=1)
```

- 最后对剩下的两类 `lang` 和 `translator_type` 直接使用 `pd.get_dummies()` 进行 one-hot 编码即可。注意需要补全的数据缺失了部分语言，需要保证训练数据和需要补全的数据有相同维度。另外 `test.json` 中同时存在 `en-GB` 和 `en-gb`，应该指同一种语言，因此将字符串转为小写。

```
def dummy2(a: pd.DataFrame, b: pd.DataFrame):
    N = len(a)
    concat_dummy = pd.get_dummies(pd.concat([a, b]))
    return concat_dummy[:N], concat_dummy[N:]
```

数据分析与更多预处理

对离散取值的属性我们不做分析，只关注正负例的数量：

```
df_label.value_counts()
```

得到有 1130 个人类数据，856 个机器人数据，正负样本没有过于失衡。尽管如此，之后也可以尝试在训练集上使用 SMOTE 等技术。

接下来分析几个连续取值属性的特征。

- 首先使用 `describe` 观察均值、方差、分位数等基本特征。从其中就可以看出，`count` 相关的属性都存在严重的不均衡，对它们进行 log 修正以消除长尾。但对于最后两个属性又出现了一定的矫枉过正，对它们再进行不同程度的修正。
- 使用 `seaborn` 画出直方图，可以看到问题得到了一定的改善。同时，从直方图中可以看到人类和机器人数据的分布存在一定的差别。

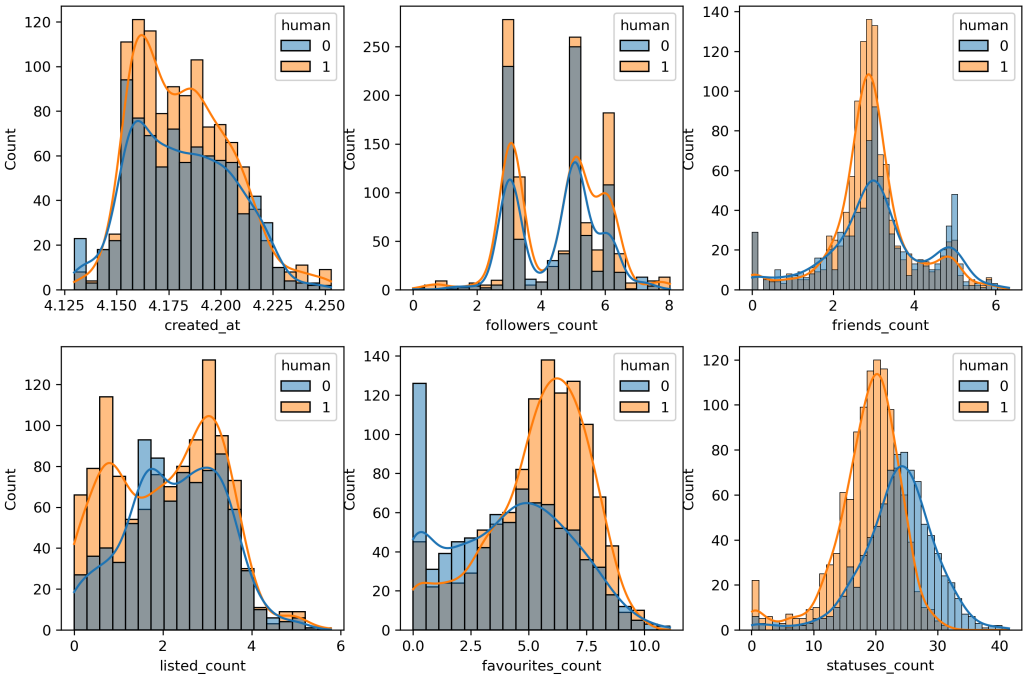

Figure 1: 几个连续属性数据的直方图

- 画出相关性热图，可以看出 `count` 相关的域均有不同程度的正相关，而创建时间与其它 `count` 相关的域有负相关。（因此，不应该直接使用朴素贝叶斯分类器等假定独立的模型）

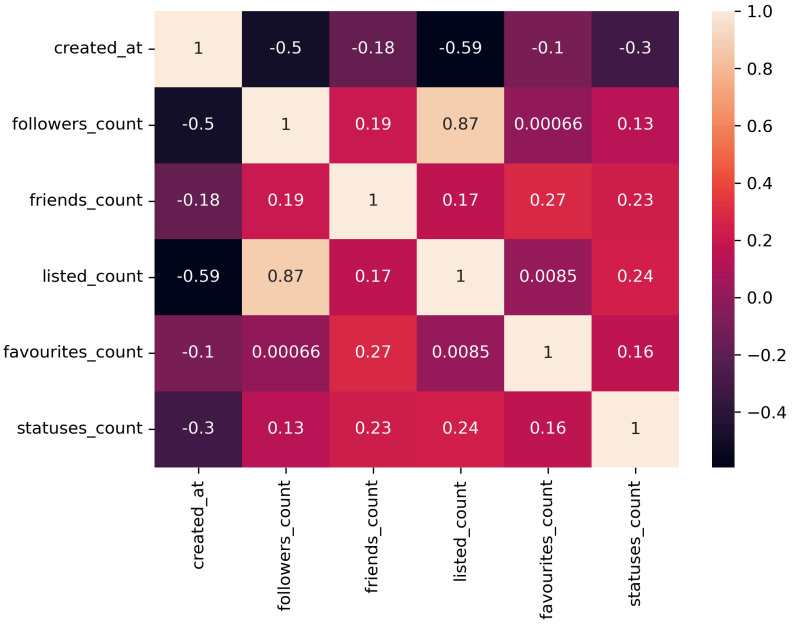

Figure 2: 几个连续属性数据的相关性图

数据集的拆分使用 `sklearn` 中的 `train_test_split` 即可。拆分后，我们对训练集单独做归一化，并对测试集和验证集做相同变换（不泄露测试集的任何数据）。去除离群值等操作也应当对训练集单独进行。

```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train.values)
X_test = scaler.transform(X_test.values)
df2fill_scaled = scaler.transform(df2fill.values)
```

模型训练

使用的模型有逻辑回归、支持向量机、随机森林、梯度提升树等（还尝试过一些其它模型，但由于效果较差没有放进来比较）。这些模型在课上都花了较大篇幅介绍，因此不再介绍原理。所有模型均直接使用 `sklearn` 等第三方库，因为它们都是较为广泛使用、传播的模型，没有必要抛弃已经很成熟的实现而另外造轮子（另外前两种模型在之前的实验也已经实现过了）。

`sklearn` 提供的这些算法一般来说关键参数不多，可以直接使用网格法选择参数。对已经调节参数后的模型进行比较，随机划分数据集 5 次，比较它们的平均准确率。一次运行得到的数据如下：

	Model	Accuracy
2	Random Forest	77.889447
0	Logistic Regression	77.336683
1	Linear SVC	77.336683
3	Gradient Boosting	77.135678

Figure 3: 几个经典模型的平均准确率比较

随机森林算法的一般表现更好，这符合我们的期望。

引入更多信息

尝试使用一些其它技巧，但模型的表现已经难以提升。这可能是因为我们之前丢弃的数据太多了，抛弃了许多有用的信息，考虑重新加入一部分。`entities` 域仍然难以解析，因此考虑加入字符串和 URL 数据。

字符串数据包括用户名、所在地、简述等信息，解析它们需要使用一些自然语言处理技术。而对于 URL 数据，有用户提供的个人链接和背景、头像等图片的链接两种类型。对前者我们不期望它能包含很多有用的数据；而对后者可以尝试使用一些 CV 技术处理。遗憾的是，图片以链接的方式给出，而其中相当一部分已经不能正常访问了（即缺失值过多）。因此，我们暂时只尝试引入一些语义信息。

简单观察数据。机器人账号的用户名可能更倾向于随机编造，因此首先两个启发式的方法是添加字符串的长度和信息熵两个域。若要引入更多语义信息，可以采用预训练模型获取字符串的嵌入表示。但这份数据从推特获得，语料质量过低，且包括多种语言，因此不能期望提取很有用的语义信息，而只是降维后保留极少一部分维度。

```
def str_embedding(s: str, max_len: int):
    text_dict = tokenizer(
        s, max_length=max_len, truncation=True,
        padding="max_length", return_tensors="pt",
    )
    return pd.Series(
        model(text_dict["input_ids"], attention_mask=text_dict["attention_mask"])[0]
        .detach().squeeze(0)
        .numpy().flatten()
    )
de = d[name].apply(str_embedding, args=(mxlen,))
pca = PCA(n_components=pca_dim)
de = pd.DataFrame(pca.fit_transform(de))
de.columns = list(map(lambda x: name + str(x), range(pca_dim)))
d = pd.concat([d, de], axis=1)
```

模型验证

选择表现最好的随机森林算法。将验证集的预测数据填入 `json` 文件中即可。

```
with open("./data/rawtest.json", "r") as file:
    data = json.load(file)
    for i in range(len(data)):
        data[i]["label"] = "human" if pred[i] else "bot"

with open("./data/test.json", "w") as file:
    json.dump(data, file, indent=4)
```