DAM Assignment2

吴桐欣 1652677

数据预处理

直接使用`dianping_user_vec.csv`文件。

用字典存储user信息，使用户特征向量和用户的性别标签对应起来。

```
    users = {}
    with open('./data/dianping_gender.csv', 'r') as file:
        for row in csv.reader(file):
            users[row[1]] = {'gender': row[2]}
    with open('./data/dianping_features/dianping_user_vec.csv', 'r') as
file:
        for row in csv.reader(file):
            if users.get(row[1]):
                elements = row[2].strip('[] ')
                elements = ' '.join(elements.split()).split(' ')
                elements = list(map(float, elements))
                users.get(row[1])['vec'] = elements
```

数据集划分

用8:1:1的比例划分训练集、验证集和测试集

调参时，根据验证集的结果，选出效果最好的一组参数

最终以测试集的结果代表该分类器的效果

```
    def split_dataset(self):
        self.train_features, self.test_features, self.train_labels, self.test_labels = \
            train_test_split(self.features, self.labels, test_size=0.2, random_state=0)
        self.validation_features, self.test_features, self.validation_labels, self.test_labels = \
            train_test_split(self.test_features, self.test_labels, test_size=0.5, random_state=0)
```

分类器

1.Logistic Regression

正则化

> 知乎·机器学习中常常提到的正则化到底是什么意思
>
> 知乎·0 范数、1 范数、2 范数有什么区别

正则化是用来防止过拟合的。 拟合是用多项式函数来进行拟合。过拟合的时候，项会很多，意味着涉及的特征比较多(而某些特征可能其实是可以忽略的)，因此要控制项的数量——同时让拟合的偏差和项数达到尽量小。拟合偏差过大，说明欠拟合；拟合偏差很小，而项数很大，说明过拟合。 如何使项数最小化？涉及范数的概念。

- 0范数：向量中非零元素的个数
- 1范数：绝对值之和
- 2范数：就是通常意义上的(向量的)模(平方和开根号)

或许可以这样简单理解：假设有向量[a1，a2，...，an]，如：[1，2，3]

范数	[a1，a2，...，an]	[1，2，3]
0范数	= n	= 3
1范数	= |a1|+|a2|+...+|an|	= 6
2范数	= √{(a1)^2 + (a2)^2 +...+ (an)^2}	= √14

要使项数最小化，则是令0范数最小，但0范数不好求，所以转而求1范数、2范数

> L2范数是指向量各元素的平方和然后求平方根。我们让L2范数的正则项||W||2最小，可以使得W的每个元素都很小，都接近于0，但与L1范数不同，它不会让它等于0，而是接近于0，这里是有很大的区别的哦；所以大家比起1范数，更钟爱2范数。

参数

参数	类型	默认值	可选项	说明
penalty	str	‘l2’	‘l1’ 或 ‘l2’	**惩罚项**，即正则化项，该项对模型向量进行“惩罚”，从而避免单纯最小二乘问题的过拟合问题。这里选择使用'l1'范数还是'l2'范数进行计算"惩罚"。一般用'l2'就够了。
C	float	1.0	正的浮点数	**正则化强度的倒数**，数字越小，则正则化的强度越大 参考
fit_intercept	bool	True	True or False	**模型的截距**，如果是False，则截距为0
class_weight	dict or ‘balanced’	None	dict or ‘balanced’	**分类的权重**，若为balanced，则根据样本中类别出现的比例自动调整分类权重
random_state	int	None	RandomState instance or None or int	生成随机数的种子，或随机数的生成器
solver	str	‘liblinear’	{‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’}	**优化算法**，决定了我们对逻辑回归损失函数的优化方法。
multi_class	str	‘ovr’	{‘ovr’, ‘multinomial’, ‘auto’}	**类的数量**，One vs Rest

> sklearn逻辑回归类库使用小结
>
> sklearn文档

结果

数据集	class_weight	solver	accuracy	precision	recall	f1
validation	None	liblinear	0.831	0.8318	0.962	0.8922
	balanced	liblinear	0.7983	0.9015	0.8113	0.854
	None	newton-cg	0.8326	0.8328	0.9632	0.8933
	balanced	newton-cg	0.795	0.899	0.809	0.8516
	None	lbfgs	0.831	0.8325	0.9609	0.8921
	balanced	lbfgs	0.7941	0.8988	0.8078	0.8509
	None	sag	0.8326	0.8328	0.9632	0.8933
	balanced	sag	0.795	0.899	0.809	0.8516
	None	saga	0.8326	0.8328	0.9632	0.8933
	balanced	saga	0.795	0.899	0.809	0.8516
test	None	newton-cg	0.8209	0.838	0.9416	0.8868

2.SVM

参数

参数	类型	默认值	可选项	说明
kernel	str	'rbf'	‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’ or a callable	核函数有四种内置选择，‘linear’即线性核函数, ‘poly’即多项式核函数, ‘rbf’即高斯核函数, ‘sigmoid’即sigmoid核函数。如果选择了这些核函数， 对应的核函数参数在后面有单独的参数需要调 参考
degree	int	3		如果我们在kernel参数使用了多项式核函数 'poly'，那么我们就需要对这个参数进行调参
gamma	float	'auto_deprecated'	'scale', 'auto', 'auto_deprecated' or float	如果我们在kernel参数使用了多项式核函数 'poly'，高斯核函数‘rbf’, 或者sigmoid核函数，那么我们就需要对这个参数进行调参
coef0	float	0.0		如果我们在kernel参数使用了 'poly'或'sigmoid'，那么我们就需要对这个参数进行调参
class_weight	dict or ‘balanced’	None	dict or ‘balanced’	**分类的权重**，若为balanced，则根据样本中类别出现的比例自动调整分类权重
decision_function_shape	str	'ovr'	'ovo', 'ovr'	OvR(one vs rest)，OvO(one vs one)
cache_size	float	200(MB)		在大样本的时候，缓存大小会影响训练速度，因此如果机器内存大，推荐用500MB甚至1000MB

> SVM调参策略
>
> sklearn文档

结果

参数	accuracy	precision	recall	f1
decision_function_shape='ovr'	0.7464	0.746	1.0	0.8545
decision_function_shape='ovo'	0.7464	0.746	1.0	0.8545
class_weight = 'balanced'	0.7556	0.8747	0.7843	0.827

核函数调参

数据集	核函数	参数	accuracy	precision	recall	f1
validation	linear		0.8377	0.8433	0.954	0.8952
	rbf	gamma='auto'	0.7272	0.7272	1.0	0.8421
	rbf	gamma='scale'	0.7916	0.7919	0.9678	0.8711
	sigmoid	gamma='auto'	0.7272	0.7272	1.0	0.8421
	sigmoid	gamma='scale'	0.6987	0.774	0.8274	0.7998
	poly	degree=1, gamma='scale'	0.7774	0.7743	0.9793	0.8648
	poly	degree=2, gamma='scale'	0.7891	0.787	0.9735	0.8704
	poly	degree=3, gamma='scale'	0.7908	0.789	0.9724	0.8711
	poly	degree=4, gamma='scale'	0.7883	0.7868	0.9724	0.8698
	poly	degree=5, gamma='scale'	0.7933	0.7917	0.9712	0.8724
	poly	degree=6, gamma='scale'	0.7849	0.7849	0.9701	0.8677
	poly	degree=7, gamma='scale'	0.7874	0.786	0.9724	0.8693
	poly	degree=8, gamma='scale'	0.7858	0.7851	0.9712	0.8683
	poly	degree=9, gamma='scale'	0.7833	0.784	0.9689	0.8667
	poly	degree=10, gamma='scale'	0.7833	0.7835	0.9701	0.8668
	poly	degree=11, gamma='scale'	0.7866	0.7858	0.9712	0.8688
test	linear		0.8209	0.838	0.9416	0.8868

3.Random Forest

参数

参数	类型	默认值	可选项	说明
n_estimators	int	10		**森林中决策树的数量**。n_estimators太小，容易欠拟合，n_estimators太大，计算量会太大
max_depth	int	None		**树的最大深度**，如果不输入的话，决策树在建立子树的时候不会限制子树的深度
min_samples_split	int, float	2		**分解内部结点时所需的最少样本数**，限制了子树继续划分的条件。样本量数量级非常大时推荐增大这个值
min_samples_leaf	int, float	1		**叶结点的最小样本数**，如果某叶子节点数目小于样本数，则会和兄弟节点一起被剪枝。样本量数量级非常大时推荐增大这个值
min_weight_fraction_leaf	float	0.0		**叶子节点最小的样本权重和**，限制了叶子节点所有样本权重和的最小值，如果小于这个值，则会和兄弟节点一起被剪枝。如果设置为0，则表示不考虑权重
max_features	int, float, string or None	'auto'	'auto','sqrt','log2'	**决策使用的最大特征数量**，如果是整数，代表考虑的特征绝对数。如果是浮点数，代表考虑特征百分比，即考虑（百分比*N）取整后的特征数
max_leaf_nodes	int or None	None		**最大叶子节点数**，可以防止过拟合。如果加了限制，算法会建立在最大叶子节点数内最优的决策树。默认None，即不限制最大叶子节点数
min_impurity_split	float	1e-7		**节点划分最小不纯度**，限制了决策树的增长，如果某节点的不纯度(基于基尼系数，均方差)小于这个阈值，则该节点不再生成子节点
oob_score	bool	False	True or False	是否采用袋外样本来评估模型的好坏

> [scikit-learn随机森林调参小结]
>
> [sklearn文档]

结果

(调参数据只选取部分，不一一列出)

数据集	固定参数	变化参数	accuracy	precision	recall	f1
validation	默认参数	默认参数	0.764	0.8139	0.8757	0.8437
	oob_score=True	n_estimators=10	0.764	0.8139	0.8757	0.8437
		n_estimators=20	0.7791	0.809	0.9114	0.8571
		n_estimators=30	0.79	0.8096	0.9298	0.8656
		n_estimators=40	0.7858	0.805	0.931	0.8634
		n_estimators=50	0.7816	0.798	0.9367	0.8618
		n_estimators=60	0.7891	0.8027	0.9413	0.8665
		n_estimators=70	0.7916	0.8045	0.9425	0.868
		n_estimators=80	0.7883	0.8002	0.9448	0.8665
		n_estimators=90	0.7824	0.7965	0.9413	0.8629
	oob_score=True, n_estimators=70	max_depth=17, min_samples_split=7	0.7816	0.7901	0.9528	0.8638
		max_depth=17, min_samples_split=8	0.7782	0.7876	0.9517	0.8619
		max_depth=17, min_samples_split=9	0.7824	0.7931	0.9482	0.8637
		max_depth=19, min_samples_split=2	0.7908	0.8014	0.9471	0.8681
		max_depth=19, min_samples_split=3	0.7808	0.791	0.9494	0.863
		max_depth=19, min_samples_split=4	0.7699	0.7829	0.9459	0.8567
	oob_score=True, n_estimators=70, max_depth=19	max_features=12	0.7824	0.7903	0.954	0.8644
		max_features=13	0.7808	0.7921	0.9471	0.8627
		max_features=14	0.7908	0.8014	0.9471	0.8681
		max_features=15	0.7724	0.7918	0.9321	0.8562
		max_features=16	0.7715	0.7876	0.939	0.8567
		max_features=17	0.7791	0.7923	0.9436	0.8613
test	oob_score=True, n_estimators=70, max_depth=19, max_features=14		0.7858	0.812	0.927	0.8657

4.Newral Network

参数

```
model = keras.Sequential([
            keras.layers.Dense(128, activation=tf.nn.relu),
            keras.layers.Dense(32, activation=tf.nn.relu),
            keras.layers.Dense(2, activation=tf.nn.softmax)
        ])
        model.compile(optimizer=tf.train.AdamOptimizer(),
                      loss=keras.losses.sparse_categorical_crossentropy,
                      metrics=['accuracy'])
```

数据集	accuracy	precision	recall	f1
validation	0.8234	0.8615	0.9022	0.8814
test	0.8176	0.8733	0.8831	0.8782

5.所有分类器比较

分类器	accuracy	precision	recall	f1
LR	0.8209	0.838	0.9416	0.8868
SVM	0.8209	0.838	0.9416	0.8868
RF	0.7858	0.812	0.927	0.8657
NN	0.8176	0.8733	0.8831	0.8782

逻辑回归和支持向量机的效果比较好。神经网络反而效果不是那么好。

总的来说，主要因为这次的用户性别预测数据集本身就不需要太复杂的模型，线性的预测就已经有比较好的结果，如神经网络多加几层就容易过拟合，因此导致逻辑回归分类器与SVM中使用'linear'参数时的分类器表现最好。