山东大学

Data Mining

—— Adult

姓　　名 陶睿

学　　号 201714738

学　　院 计算机科学与技术

专　　业 计算机科学

年　　级 2017 级

目 录

一、 Adult 数据集 1
1.1. 数据简介 1
1.2. 数据简要分析 2
二、 数据预处理 2
2.1. 实验环境 2
2.2. 转化为 CSV 格式 2
2.3. 移除缺失值 3
2.4. 离散型转为数值型 4
2.5. 特征缩放和 PCA 降维 6
三、 分类器模型 8
3.1. 核 SVM 8
3.2. 贝叶斯分类器 BNB 8
3.3. 随机梯度递减 SGD 9
3.4. SVM、BNB、SGD 组合分类器 9
3.5. DNN、Linear 组合分类器 11
四、 实验结果及分析 13
4.1. PCA 降维的维度选择对时间和准确率的影响 13
4.2. 训练样本个数对时间和准确率的影响 13
4.3. SVM、BNB、SGD 及其组合的评估 14
4.4. DNNLinear 组合分类器与其他三种的对比 15
五、 结论 16

一、Adult 数据集

1.1.数据简介

该数据从美国 1994 年人口普查数据库中抽取而来，因此也称作“人口普查收入”数据集，训练数据 32561 条和测试数据 16281 条。该数据集类变量为年收入是否超过 50k$，是一个分类数据集，用来预测年收入是否超过 50k$。

数据的属性见表 1.1.1：

表 1.1.1 数据属性表

属性名	类型	含义
age	continuous	年龄
workclass	discrete	工作类别
fnlwgt	continuous	序号
education	discrete	受教育程度
education-num	continuous	受教育时间
marital-status	discrete	婚姻状况
occupation	discrete	职业
relationship	discrete	社会角色
race	discrete	种族
sex	discrete	性别
capital-gain	continuous	资本收益
capital-loss	continuous	资本支出
hours-per-week	continuous	每周工作时间
native-country	discrete	国籍

用训练数据训练的分类器，对测试数据进行测试。计算准确率（Precision），精度（Accuracy），召回率（Recall）等评估分类器。

在本次实验过程中，主要还进行了各种调参来对分类器进行提升。

通过本课题的研究，来达到以下两个目的：

1. 对数据分析课程内容的加深理解，例如数据预处理，算法选择，数据分析方法等等知识进行巩固。
2. 从实验中得到的一些数据处理、分类器的选择和调参等经验，进行归纳总结，为以后的科研需要做准备。

1.2.数据简要分析

1.2.1 年收入与学习的关系

表 1.2.1 年收入与学历的关系

学历	年收入>50K比例
博士	75%
硕士	57%
学士	42%

从表 1.2.1 可以看出，学历越高，年收入大于 50K 的比例越高。

1.2.2 年龄分布

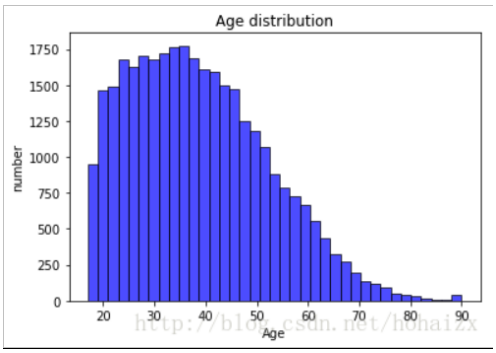


图 1.2.1 年龄分布

从图 1.2.1 年龄分布上，数据集人数最多的年龄段为 20-40 岁。

1.2.3 受教育年限分布

从图 1.2.2 教育年限上看，大部分人受教育年限为 5-10 年；约三分之一为 10 年以上，仅有不到 5%的人受教育年限不足 5 年

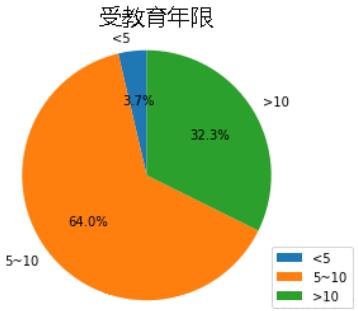


图 1.2.2 教育年限分布

1.2.4 周工作时间分布

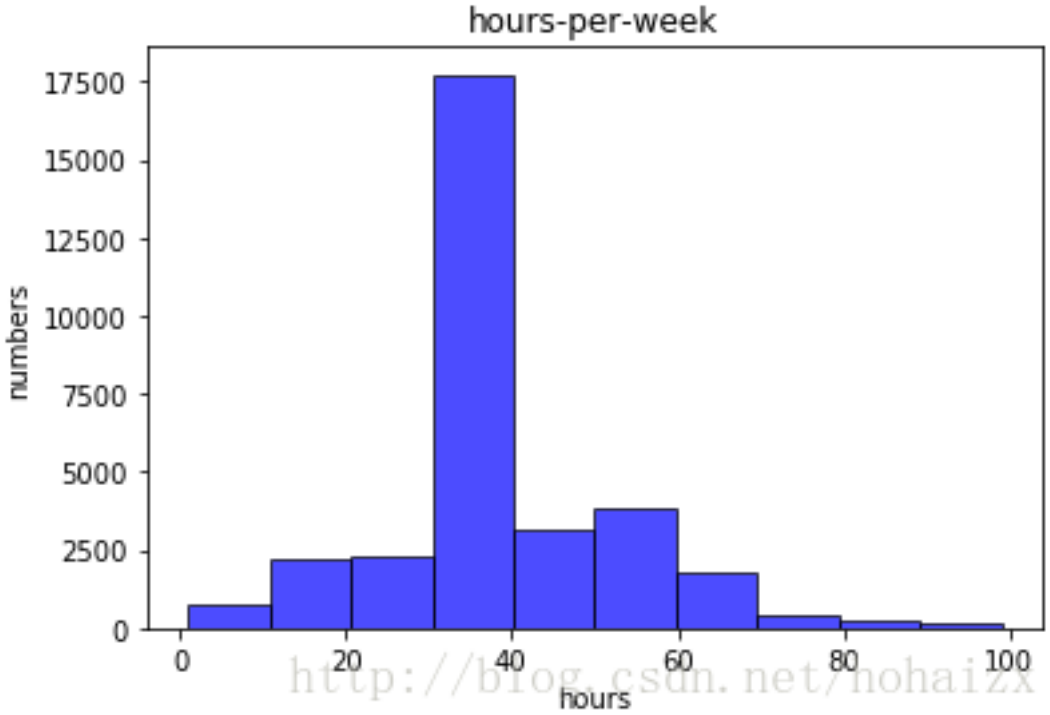


图 1.2.3 周工作时间分布

- 可以看到，大多数美国人每周工作时间集中在 30—40 小时。工作时间少于 30 小时和工作时间多余 40 小时都属于少部分人群。

- 工作时间最长的人群达到了每周 100 个小时，相当于每周工作 7 天，每天 14 个小时。

二、数据预处理

数据的质量和包含的有用信息量是决定一个机器学习算法能够学多好的关键因素。因此，我们在训练模型前评估和预处理数据显得至关重要。本次实验主要用到了一下几个方法：移除数据集中的缺失值、将分类(category)数据转型，能够被机器学习算法处理、特征选择

2.1.实验环境

- **Windows 10**
- **Python 2.7.15 & Python3.6.3**
- **Numpy 1.14.5**
- **Pandas 0.23.1**
- **Scikit-learn 0.19.1**
- **Matplotlib 2.2.2**
- **Tensorflow 1.8.0**

2.2.转化为 CSV 格式

为了更加容易去读取数据，我先将两个文件利用命令行转换成了 CSV 格式文件

```
cat adult.data  >  data.csv
cat adult.test   >  test.csv
```

得到的结果如下图所示：

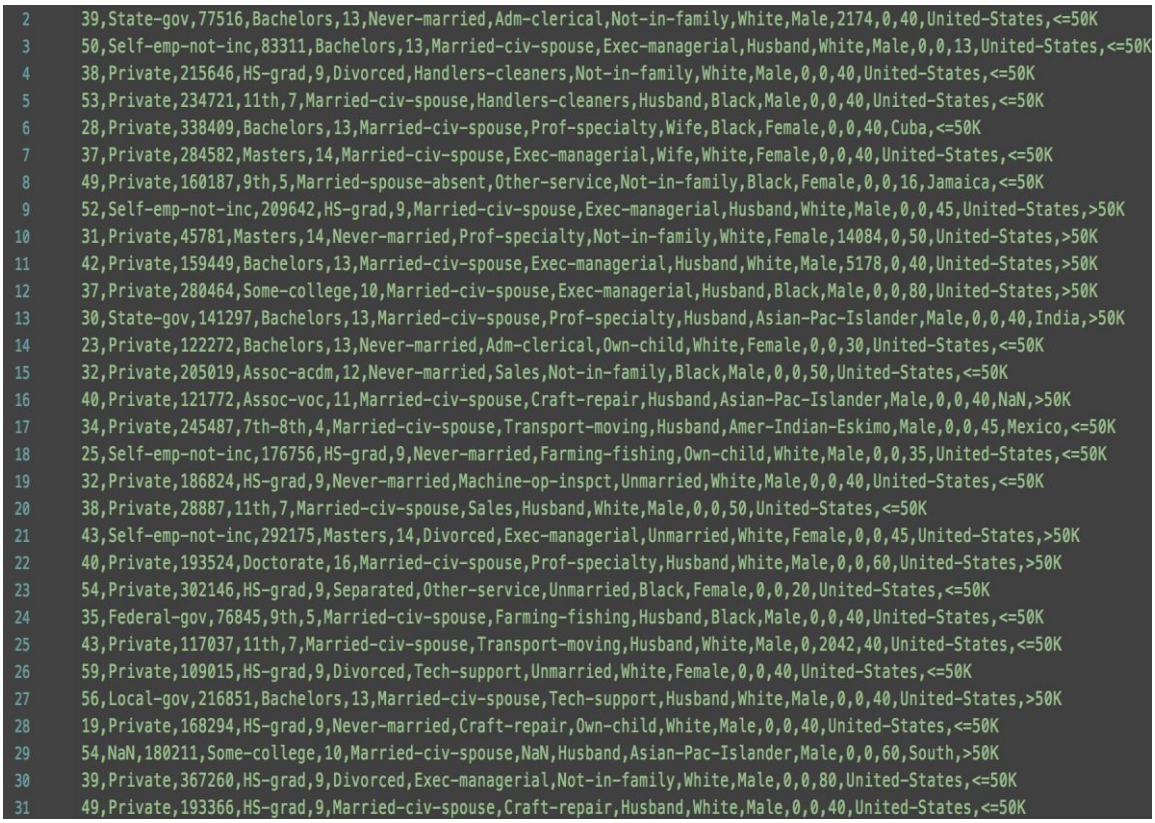


图 2.2.1 转换后的训练数据

我们得这两个文件之后，可以从中看到每条数据有 14 个属性，其中是 continues 的属性有{age, fnlwgt, education-num, capital-gain, capital-loss, hours-per-week}，枚举型离散属性有{workclass, education, marital-status, occupation, relation, race, sex, native-country}。

接下来，我们就要进行基本的数据预处理。

2.3.移除缺失值

现实中的数据总是存在或多或少的缺失值现象。原因多种多样，可能是数据收集阶段发生错误，也可能数据调研阶段某些选项没有被填写。不论是什么原因造成的缺失值，我们都统一将其看做空格或者用 NaN(Not a Number)表示的占位符。

不幸地是，大多数计算工具不能处理缺失值，即使我们忽略缺失值也不能产生预测结果。因此，我们必须认真对待缺失值问题。

```
trainData=trainData.dropna()
testData=testData.dropna()
```

图 2.3.1 移除缺失值

处理缺失值最简单的手段无疑是直接将带有缺失值的特征(列)或样本(行)从数据集中去掉。去掉行可以通过 dropna 方法：

至此，我们可以将数据集中带有“？”的删除掉。

2.4. 离散型转为数值型

接下来，我们要对数据中的属性进行转换。例如像性别(sex),工作类别(workclass),教育水平(education)等等这种无需离散型变量，我们需要把他们映射到数值型变脸。

例如性别，我们创建一个映射表，如下图：

```
#map 性别
sex_mapping={
    'Male':0,
    'Female':1
}
```

图 2.4.1 映射表

之后对两个样本集进行转换

```
trainData['sex']=trainData['sex'].map(sex_mapping)
testData['sex']=testData['sex'].map(sex_mapping)
```

图 2.4.2 性别转换

其余的属性，我们试着直接采用了第三方库提供的函数进行转换，如下图：

get_dummies 默认会对 DataFrame 中所有字符串类型的列进行 onehot 编码

```
trainData=pd.get_dummies(trainData,sparse=False)
testData=pd.get_dummies(testData,sparse=False)
```

图 2.4.3 对字符串类型进行 onehot 编码

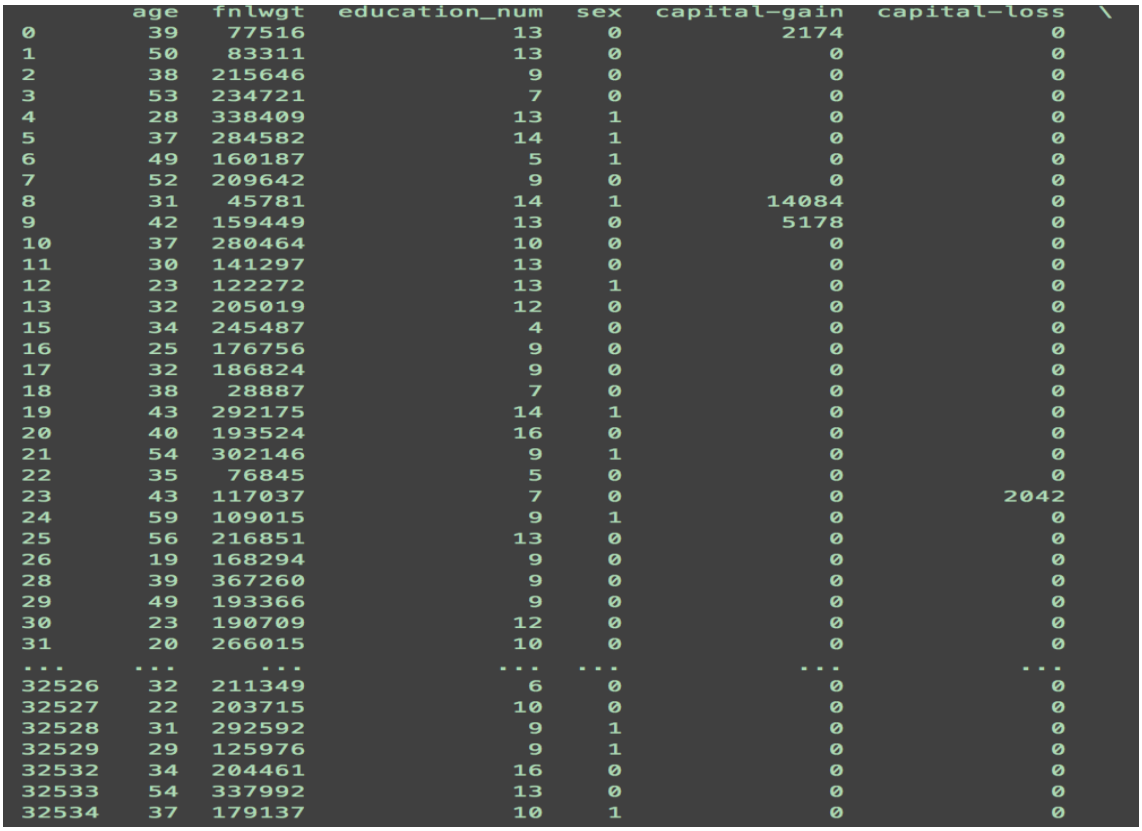
get_dummies 默认会对 DataFrame 中所有字符串类型的列进行 onehot 编码。接下来，我们需要对分类的类别进行映射，即以工资收入做分类的依据：

```
#map工资收入
train_income_mapping={
    '<=50K':0,
    '>50K':1
}
test_income_mapping={
    '<=50K.':0,
    '>50K.':1
}
```

```
trainData['income']=trainData['income'].map(train_income_mapping)
testData['income']=testData['income'].map(test_income_mapping)
```

图 2.4.3&图 2.4.4 对类别进行 onehot 编码

这样，我们就得到了一个无需离散型的数值训练样本，下图给出了部分结果的展示：



	age	fnlwgt	education_num	sex	capital-gain	capital-loss	\
0	39	77516	13	0	2174	0	
1	50	83311	13	0	0	0	
2	38	215646	9	0	0	0	
3	53	234721	7	0	0	0	
4	28	338409	13	1	0	0	
5	37	284582	14	1	0	0	
6	49	160187	5	1	0	0	
7	52	209642	9	0	0	0	
8	31	45781	14	1	14084	0	
9	42	159449	13	0	5178	0	
10	37	280464	10	0	0	0	
11	30	141297	13	0	0	0	
12	23	122272	13	1	0	0	
13	32	205019	12	0	0	0	
15	34	245487	4	0	0	0	
16	25	176756	9	0	0	0	
17	32	186824	9	0	0	0	
18	38	28887	7	0	0	0	
19	43	292175	14	1	0	0	
20	40	193524	16	0	0	0	
21	54	302146	9	1	0	0	
22	35	76845	5	0	0	0	
23	43	117037	7	0	0	2042	
24	59	109015	9	1	0	0	
25	56	216851	13	0	0	0	
26	19	168294	9	0	0	0	
28	39	367260	9	0	0	0	
29	49	193366	9	0	0	0	
30	23	190709	12	0	0	0	
31	20	266015	10	0	0	0	
...	...	...	...	...	...	...	
32526	32	211349	6	0	0	0	
32527	22	203715	10	0	0	0	
32528	31	292592	9	1	0	0	
32529	29	125976	9	1	0	0	
32532	34	204461	16	0	0	0	
32533	54	337992	13	0	0	0	
32534	37	179137	10	1	0	0	

图 2.4.5 某个编码后的训练样本

2.5.特征缩放和 PCA 降维

特征缩放(feature scaling)是预处理阶段的关键步骤，但常常被遗忘。虽然存在决策树和随机森林这种是少数不需要特征缩放的机器学习算法，但对于大部分机器学习算法和优化算法来说，如果特征都在同一范围内，会获得更好的结果

类似于特征选择，我们可以使用特征抽取来减小数据集中的特征维度。不过，不同于特征选择算法会保留原始特征空间，特征抽取会将原始特征转换/映射到一个新的特征空间。换句话说，特征抽取可以理解为是一种数据压缩的手段，同时保留大部分相关信息。特征抽取是用于提高计算效率的典型手段，另一个好处是也能够减小维度诅咒(curse of dimensionality)，特别是对于没有正则化的模型。

PCA(principal component analysis, 主成分分析)是一种被广泛使用的无监督的线性转换技术，主要用于降维。其他领域的应用还包括探索数据分析和股票交易的信号去噪，基因数据分析和基因表达。

PCA 根据特征之间的相关性帮助我们确定数据中存在的模式。简而言之，PCA 的目标是找到高维数据中最大方差的方向，并且将高维数据映射到一个新的子空间，这个子空间的方向不大于原始特征空间。新子空间的正交轴(主成分)可以被解释为原始空间的最大方差方向。

代码实现如下所示：

```
#标准化   分类结果从75%提升到了80%以上
train_x_std=StandardScaler().fit_transform(train_x)
test_x_std=StandardScaler().fit_transform(test_x)

#PCA降维
pca=PCA(n_components=2)
train_x_pca=pca.fit_transform(train_x_std)
test_x_pca=pca.transform(test_x_std)
```

图 2.5.1 标准化和 PCA 降维代码

将多个属性降维到一个二维数组，结果如下：

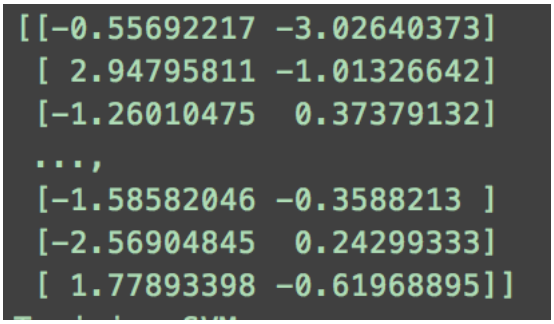

[[-0.55692217 -3.02640373]
[2.94795811 -1.01326642]
[-1.26010475 0.37379132]
...,
[-1.58582046 -0.3588213]
[-2.56904845 0.24299333]
[1.77893398 -0.61968895]]

图 2.5.2 降维后的数据

三、分类器模型

3.1. 核 SVM

SVM 之所以受欢迎度这么高，另一个重要的原因是它很容易核化(kernelized)，能够解决非线性分类问题。

核方法的 idea 是为了解决线性不可分数据，在原来特征基础上创造出非线性的组合，然后利用映射函数$\phi(\cdot)$将现有特征维度映射到更高维的特征空间，并且这个高维度特征空间能够使得原来线性不可分数据变成了线性可分的。

我们在这里选用核'rbf'，gamma = 1,代码如下：

```
print ('Training SVM')
clf_svm=svm.SVC(kernel='rbf',gamma=1.0,C=3.0)
clf_svm.fit(train_x_pca,train_y)
print ('Testing SVM')
predict_svm=clf_svm.predict(test_x_pca)
```

图 3.1.1 SVM 代码

3.2. 贝叶斯分类器 BNB

在这里，我们使用高斯贝叶斯分类器进行训练，参数使用默认

```
print ('Training BNB')
clf_bnb=GaussianNB()
clf_bnb.fit(train_x_pca,train_y)
print ('Testing BNB')
predict_bnb=clf_bnb.predict(test_x_pca)
```

图 3.2.1 BNB 代码

3.3.随机梯度递减 SGD

随机梯度下降法(stochastic gradient descent)。有时也被称为迭代(iteration)/在线(on-line)梯度下降。随机梯度下降法每次只用一个样本对权重进行更新。

虽然随机梯度下降被当作是梯度下降的近似算法，但实际上她往往比梯度下降收敛更快，因为相同时间内她对权重更新的更频繁。由于单个样本得到的损失函数相对于用整个训练集得到的损失函数具有随机性，反而会有助于随机梯度下降算法避免陷入局部最小点。

```
print ('Training SGD')
clf_SGD=SGDClassifier(penalty='l1')
clf_SGD.fit(train_x_pca,train_y)
print ('Testing SGD')
predict_SGD=clf_SGD.predict(test_x_pca)
```

图 3.3.1 SGD 代码

为了降低过拟合的风险，我们使用了'l1'正则化。

3.4.SVM、BNB、SGD 组合分类器

为了提高分类结果，我们将上面三个分类器进行组合，然后投票。

代码的实现如下图：

```
for i in range(0,row):

        count_zero=0
        count_one=0
        count_two=0
        count_three=0

        for j in range(0,col):

            if   (data.ix[i,j] == 0):
                    count_zero = count_zero + 1

            elif (data.ix[i,j]==1):
                    count_one = count_one + 1

            elif (data.ix[i,j]==2):
                    count_two = count_two + 1

            elif (data.ix[i,j]==3):
                    count_three +1

        bigger={"count_zero":count_zero, "count_one":count_one, "count_two":count_two,"count_three":count_three}
        help_max=max(bigger.iterkeys(), key=lambda k: bigger[k])

        if (help_max == "count_zero"):
            new_class.append(0)
        elif (help_max == "count_one"):
            new_class.append(1)
        elif (help_max == "count_two"):
            new_class.append(2)
        elif (help_max == "count_three"):
            new_class.append(3)
```

图 3.4.1 组合分类器代码 1

然后调用函数：

```
#组合分类器
voting=mv()
ensemble=pd.DataFrame({"svm":predict_svm,"bnb":predict_bnb,"SGD":predict_SGD})
ensemble["voting"]=voting.majority_voting(ensemble)
ensemble["original"]=list(test_y)
print ensemble
```

图 3.4.2 组合分类器代码 2

最后得到的分类器训练结果如下图所示

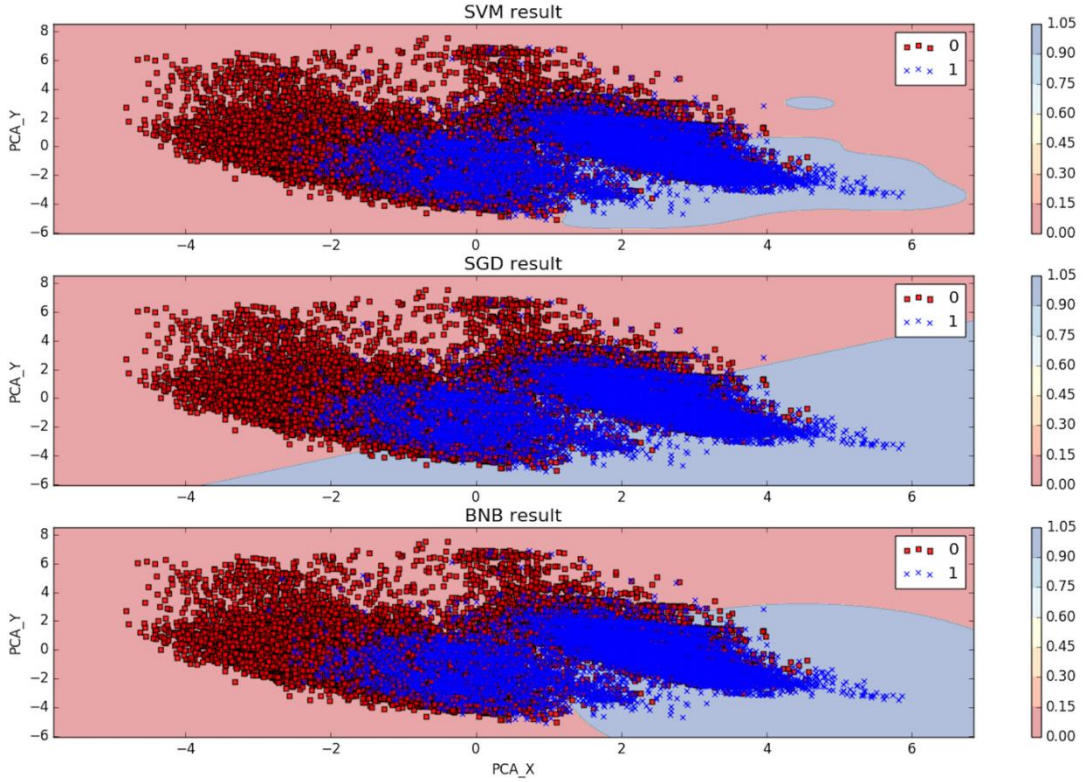


图 3.4.3 SVM、SGD、BNB 在训练集上的结果可视化

3.5.DNN、Linear 组合分类器

Linear 分类器和 Deep 神经网络分类器结合。

网络结构如图所示：

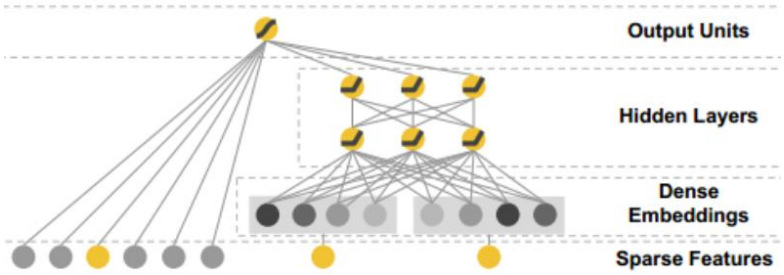


图 3.5.1 DNN 与 Linear 组合的分类器结构

实现代码如下：

```python
def build_estimator(model_dir, model_type):
  """Build an estimator."""
  if model_type == "wide":
    m = tf.estimator.LinearClassifier(
        model_dir=model_dir, feature_columns=base_columns + crossed_columns)
  elif model_type == "deep":
    m = tf.estimator.DNNClassifier(
        model_dir=model_dir,
        feature_columns=deep_columns,
        hidden_units=[100, 50])
  else:
    m = tf.estimator.DNNLinearCombinedClassifier(
        model_dir=model_dir,
        linear_feature_columns=crossed_columns,
        dnn_feature_columns=deep_columns,
        dnn_hidden_units=[100, 50])
  return m
```

图 3.5.2 DNN、Linear 组合分类器代码

四、实验结果及分析

每种参数都分别实验 3-4 次，取平均值。测试了 PCA 维度，训练数据个数的影响。并对几种不同的分类器进行了对比和评估。

4.1.PCA 降维的维度选择对时间和准确率的影响

表 4.1.1 不同 PCA 维度、训练样本数运行结果

序号	Train num	Test num	PCA	CPU TIME	SVM	Bayes	SGD	Ensemble	average
1	100	1000	4	0.52	78.30	78.17	77.67	78.90	78.26
2	1000	1000	4	0.60	80.60	80.40	68.27	80.27	77.38
3	10000	1000	4	2.50	82.80	80.63	78.07	81.83	80.83
4	20000	1000	4	7.89	83.77	82.00	77.33	82.83	81.48
5	20000	1000	8	19.68	82.00	80.13	81.10	82.15	81.34
6	20000	1000	2	15.25	82.25	81.23	80.20	81.30	81.24

- PCA 降维后可以将 14 维数据降为 dim 维数据
- 分别选取 dim 为 2 4 8
- 训练集数目 20000，测试集数目 1000
- dim 为 4 时，所用时间最少。dim>4 时，时间随 dim 增加而增加
- accuracy 上没有明显差异。

4.2.训练样本个数对时间和准确率的影响

表 4.2.1 不同 PCA 维度、训练样本数运行结果

序号	Train num	Test num	PCA	CPU TIME	SVM	Bayes	SGD	Ensemble	average
1	100	1000	4	0.52	78.30	78.17	77.67	78.90	78.26
2	1000	1000	4	0.60	80.60	80.40	68.27	80.27	77.38
3	10000	1000	4	2.50	82.80	80.63	78.07	81.83	80.83
4	20000	1000	4	7.89	83.77	82.00	77.33	82.83	81.48
5	20000	1000	8	19.68	82.00	80.13	81.10	82.15	81.34
6	20000	1000	2	15.25	82.25	81.23	80.20	81.30	81.24

- PCA 降维为 4 维数据
- 训练集数目 100/10^3/10^4/2*10^4，测试集数目 10^3
- 随着训练所用数据量增加，组合分类器 accuracy 从 78.9 提高到 82.8。
- 其中 SVM Bayes Ensemble 提高比较明显，SGD 波动比较大。

4.3.SVM、BNB、SGD 及其组合的评估

```
accuracy_svm:83.000000
accuracy_binomial naivebayes:81.100000
accuracy_SGD:80.400000
accuracy_ensemble:83.000000
svm classification
             precision    recall  f1-score   support

          0       0.86      0.93      0.89       762
          1       0.70      0.50      0.58       238

avg / total       0.82      0.83      0.82      1000

bnb classification
             precision    recall  f1-score   support

          0       0.86      0.89      0.88       762
          1       0.61      0.55      0.58       238

avg / total       0.81      0.81      0.81      1000

SGD classification
             precision    recall  f1-score   support

          0       0.81      0.98      0.88       762
          1       0.76      0.26      0.38       238

avg / total       0.80      0.80      0.76      1000

confusion matrix
[[712  50]
 [120 118]]
[[679  83]
 [106 132]]
[[743  19]
 [177  61]]
pca n = 2

Method all: 14.749609 CPU seconds
```

图 4.3.1 SVM、BNB、SGD 运行结果

三种分类器模型及其组合模型运行结果统计如下表

表 4.3.1 SVM、SGD、BNB 结果统计

	accuracy	precision	recall	f1-score
SVM	0.83	0.82	0.83	0.82
BNB	0.811	0.81	0.81	0.81
SGD	0.804	0.80	0.80	0.76
组合	0.83			

- train = 20000 test = 1000
- 四种方法准确率 Accuracy 分别为=83%、81%、80%、83%
- 对工资少于 50K 的分类结果普遍较好，对大于 50K 的分类很差。原因可能是工资大于 50K 的样本数少且分布混在在工资小于 50K 的样本中。

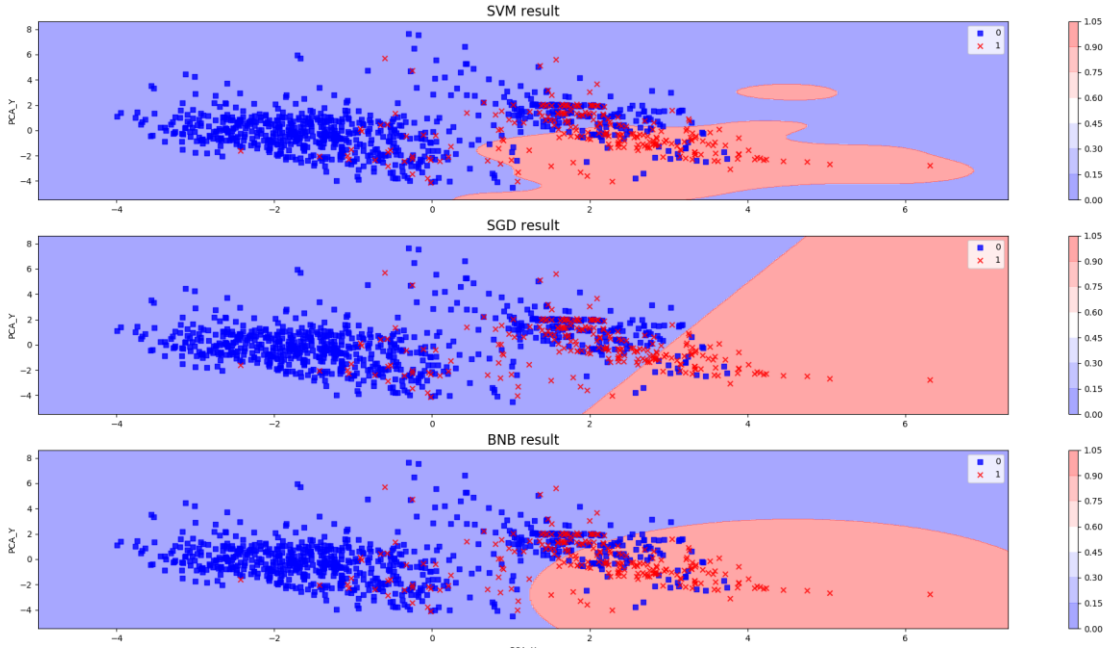


图 4.3.2 SVM、SGD、BNB 结果可视化

如上图所示，我们可以直观地看到分类器的结果，接下来我们需要得到准确的数字。

4.4. DNNLinear 组合分类器与其他三种的对比

experiment	1	2	3	4	5
train_steps	1000	5000	10000	20000	50000
Accuracy	0.805	0.841	0.846	0.85	0.853

- 迭代次数从 1000 增加到 50000，准确率 Accuracy 从 0.805 增长到 0.853
- 迭代次数 50000 运行时间过久，未跑出结果。
- 结合 DNN 和线性分类器所得的分类器，效果优于前面实验过的三种分类器，而且随着迭代步数的增加，accuracy 仍在增加。

五、结论

通过对本题目的研究，我对数据分析的了解更加深入了一步，掌握了使用支持向量机 SVM、朴素贝叶斯 BNB、随机梯度下降 SGD、线性和深度网络组合分类器等对一个数据进行分类。举一反三，聚类等等问题也可以迎刃而解。

此外，实验过程中涉及到许多调参的操作，对平时训练各种神经网络时的调参也有一定的启发作用。