# Data Science - Homework 7 (Classification)

- Introduction of datasets

## 1. Adult Dataset (from UCI)

An individual's annual income results from various factors. Intuitively, it is influenced by the individual's education level, age, gender, occupation, and etc. This dataset also known as "Census Income" dataset. Therefore, the goal of this assignment is to accurately predict whether an adult makes more than 50K in an year on the basis of the features given.

| | age | workclass | fnlwgt | education | education_num | marital_status | occupation | relationship | race | gender | capital_gain | capital_loss | hours_per_week |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **32556** | 27 | Private | 257302 | Assoc-acdm | 12 | Married-civ-spouse | Tech-support | Wife | White | Female | 0 | 0 | 38 |
| **32557** | 40 | Private | 154374 | HS-grad | 9 | Married-civ-spouse | Machine-op-inspct | Husband | White | Male | 0 | 0 | 40 |
| **32558** | 58 | Private | 151910 | HS-grad | 9 | Widowed | Adm-clerical | Unmarried | White | Female | 0 | 0 | 40 |
| **32559** | 22 | Private | 201490 | HS-grad | 9 | Never-married | Adm-clerical | Own-child | White | Male | 0 | 0 | 20 |
| **32560** | 52 | Self-emp-inc | 287927 | HS-grad | 9 | Married-civ-spouse | Exec-managerial | Wife | White | Female | 15024 | 0 | 40 |

First, after importing the dataset, I found that it has 32561 records in training set, and test set has 16282 records, both of them has 15 attributes. Next, I check the details of training set, as can be seen from the figure on the right that there are missing values in three columns, including workclass, occupation and native_country. I will drop the rows with missing values.

```
df_adult_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
age               32561 non-null int64
workclass         30725 non-null object
fnlwgt            32561 non-null int64
education         32561 non-null object
education_num     32561 non-null int64
marital_status    32561 non-null object
occupation        30718 non-null object
relationship      32561 non-null object
race              32561 non-null object
gender            32561 non-null object
capital_gain      32561 non-null int64
capital_loss      32561 non-null int64
hours_per_week    32561 non-null int64
native_country    31978 non-null object
income_bracket    32561 non-null object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

## 2. Annealing dataset (from UCI)

The exact meaning of the features and classes is largely unknown. Annealing, in metallurgy and materials science, is a heat treatment that alters the physical and sometimes chemical properties of a material to increase its ductility and reduce its hardness, making it more workable. It involves heating a material to above its recrystallization temperature, maintaining a suitable temperature, and then cooling.

- Use Python to perform classification analysis of some datasets and explain the results I obtain..

## 1. Adult Dataset (from UCI)

### Preprocessing

First, in the process of data cleaning, I removed the fnlgwt and education, because I observed that fnlgwt attribute has a lot of distinct values by getting the count of unique values, and the education and education_num this two columns have the same meaning.

Second, I need to convert categorical data to something that the machine could better understand, the one-hot encoding is a helpful method. But using one-hot encoding increases the dimensionality of the dataset, I tried to implement PCA to solve this problem. I hope that the overall performance will not vary too much or even better.

### Train model

There are three models I mainly used, including logistic regression (LR), decision tree (DT) and KNN.

In addition, in order to train the model and solve the problem of parameter adjustment, I used pipeline and GridSearch technology in each model training. My other parameter settings are shown on the right. For the logistic regression model, the parameter of GridSearch I chose 'C'. It is worth noting that the 'scoring' param is roc_auc not accuracy, because I think that the performance of classification depends on roc_auc is better than accuracy, it is more objective. The result of logistic regression is about 0.886, I believed that it is a nice result.

```
scaler = StandardScaler()
pca = PCA(n_components=40)
lr = LogisticRegression(random_state=1, solver='lbfgs')

pipe_lr = Pipeline(steps=[('scaler', scaler),
                          ('pca', pca),
                          ('lr', lr)])

param_grid = [{'lr__C':np.logspace(-3,3,7)}]

gs = GridSearchCV(estimator=pipe_lr,
                  param_grid=param_grid,
                  scoring='roc_auc',
                  cv=10)

scores = cross_val_score(gs, X_train, y_train,
                         scoring='roc_auc', cv=10)

gs = gs.fit(X_train, y_train)
```

```
0.8871662682803529
{'lr__C': 1000.0}
CV roc_auc: 0.886 +/- 0.008

Train accuracy: 0.834
Test Accuracy: 0.833
Misclassified examples: 1514
```

There are some illustrations for last result graph of the previous page, I displayed the misclassified examples, we can see that this model leads 1514 data were classified incorrectly.

```
Confusion Matrix =
[[6258  539]
 [ 975 1277]]

Classification Report =
              precision    recall  f1-score   support

           0       0.87      0.92      0.89      6797
           1       0.70      0.57      0.63      2252

    accuracy                           0.83      9049
   macro avg       0.78      0.74      0.76      9049
weighted avg       0.82      0.83      0.83      9049
```

Furthermore, I also implemented the confusion matrix and classification report for this model, we can observe that the sum of anti-diagonal in confusion matrix is exactly 1514. Let's move on to the classification report, it gives us the values of the precision, recall and f1-score respectively for two classes, the class 0 has the better performance on the classification. The only reason I can think of is the scarcity of training data in class 1, because I summarized the class distribution, where about 75% of the population has income <= 50k and only 25% > 50k, so there may be a data imbalance problem.

Lastly, I train the other two models, their experimental results and parameter settings are shown in the figure below and the next page. I will discuss them on the next page.

```
dt = DecisionTreeClassifier(random_state=0)

pipe_dt = Pipeline(steps=[('pca', pca),
                          ('dt', dt)])

param_grid = [{'dt__max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, None],
               'dt__criterion': ['gini','entropy']}]

gs = GridSearchCV(estimator=pipe_dt,
                  param_grid=param_grid,
                  scoring='roc_auc',
                  cv=10)

scores = cross_val_score(gs, X_train, y_train,
                         scoring='roc_auc', cv=10)

gs = gs.fit(X_train, y_train)
print(gs.best_score_)
print(gs.best_params_)
print('CV roc_auc: %.3f +/- %.3f' % (np.mean(scores),
                                     np.std(scores)))
```

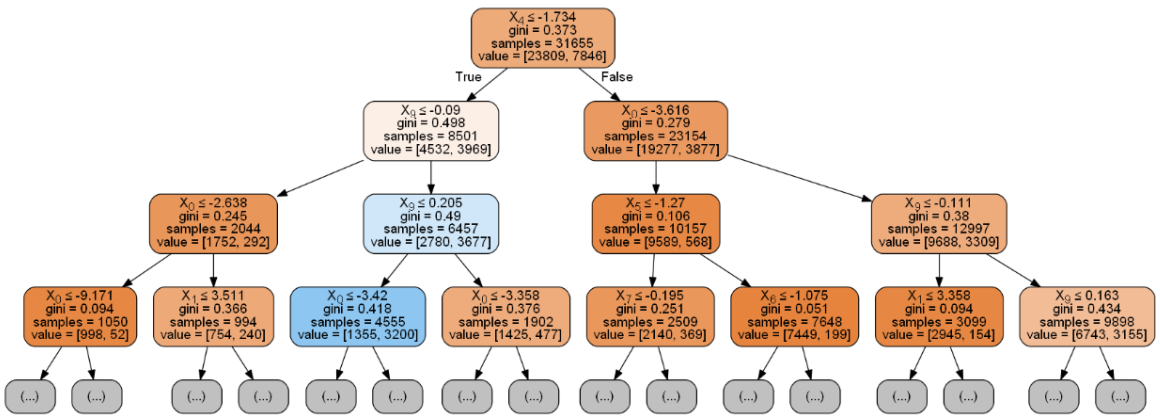
```
Confusion Matrix =
[[6410  387]
 [1016 1236]]

Classification Report =
              precision    recall  f1-score   support

           0       0.86      0.94      0.90      6797
           1       0.76      0.55      0.64      2252

    accuracy                           0.84      9049
   macro avg       0.81      0.75      0.77      9049
weighted avg       0.84      0.84      0.84      9049
```

```
0.8838548093444253
{'dt__criterion': 'gini', 'dt__max_depth': 7}
CV roc_auc: 0.883 +/- 0.007
```



```python
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(p=2, metric='minkowski')

pipe_knn = Pipeline(steps=[('pca', pca),
                           ('knn', knn)])

param_grid = [{'knn__n_neighbors': [10, 15, 20, 25, 30, 35, 40, 45, 50]}]

gs = GridSearchCV(estimator=pipe_knn,
                  param_grid=param_grid,
                  scoring='roc_auc',
                  cv=10)

scores = cross_val_score(gs, X_train, y_train,
                         scoring='roc_auc', cv=10)

gs = gs.fit(X_train, y_train)
print(gs.best_score_)
print(gs.best_params_)
print('CV roc_auc: %.3f +/- %.3f' % (np.mean(scores),
                                     np.std(scores)))
```

```
Confusion Matrix =
[[6338  459]
 [ 899 1353]]

Classification Report =
              precision    recall  f1-score   support

           0       0.88      0.93      0.90      6797
           1       0.75      0.60      0.67      2252

    accuracy                           0.85      9049
   macro avg       0.81      0.77      0.78      9049
weighted avg       0.84      0.85      0.84      9049
```

```
0.8961804031999622
{'knn__n_neighbors': 25}
CV roc_auc: 0.896 +/- 0.004
```

In the decision tree model, I selected the max-depth and criterion to search the best parameter combination for this adult dataset, and the n_neighbors are used in the KNN model. Compared with three models, I think that There are not much different in their performance, the value of roc_auc fall in 0.86~0.89, and the misclassification fall in 1300~1500 data.

## 2. Annealing dataset (from UCI)

### Preprocessing

| | family | product-type | steel | carbon | hardness | temper_rolling | condition | formability | strength | non-ageing | ... | s | p | shape | thick | width | len | oil | bore |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **793** | NaN | C | A | 0 | 50 | T | NaN | NaN | 0 | NaN | ... | NaN | NaN | COIL | 1.001 | 50.0 | 0 | Y | 0 |
| **794** | NaN | C | NaN | 0 | 0 | NaN | S | 1.0 | 0 | NaN | ... | NaN | NaN | SHEET | 0.699 | 1300.0 | 4880 | NaN | 0 |
| **795** | NaN | C | A | 0 | 0 | NaN | S | 2.0 | 0 | NaN | ... | NaN | NaN | COIL | 0.400 | 609.9 | 0 | NaN | 0 |
| **796** | NaN | C | A | 0 | 0 | NaN | S | 2.0 | 0 | NaN | ... | NaN | NaN | SHEET | 3.200 | 610.0 | 4880 | NaN | 0 |
| **797** | NaN | C | R | 0 | 0 | NaN | S | 2.0 | 0 | NaN | ... | NaN | NaN | SHEET | 1.599 | 1500.0 | 4170 | NaN | 0 |

From the above, we can see that there are a lot of meaningless columns and data, if they are the categorical data, I would drop them to let me handle the next steps better. For the numerical data, I would interpolate the values by mean. And it has a classes column, I directly use them as the labels for my classification task, the values of this column include 1, 2, 3, 4, 5 and U, then I split them to the two groups, the first group is classes=1~3, and the second group is classes=4, 5, U.

### Train models

Compared to the first dataset, I decided to use more classification models to analyze for the second dataset, and then the evaluation indicators are the same.

There are the other three models I mainly used, including random forest (RF), MajorityVoteClassifier and BaggingClassifier.

In the beginning, I wanted to introduce simply for the MajorityVoteClassifier.

Two different voting schemes are common among voting classifiers:

- In hard voting (also known as majority voting), every individual classifier votes for a class, and the majority wins. In statistical terms, the predicted target label of the ensemble is the mode of the distribution of individually predicted labels.
- In soft voting, every individual classifier provides a probability value that a specific data point belongs to a particular target class. The predictions are weighted by the classifier's importance and summed up. Then the target label with the greatest sum of weighted probabilities wins the vote.

The code for this classifier as shown below, I discussed on the next page.

```
clf1 = LogisticRegression(penalty='l2',
                          C=0.001,
                          solver='lbfgs',
                          random_state=1)

clf2 = DecisionTreeClassifier(max_depth=1,
                              criterion='entropy',
                              random_state=0)

clf3 = KNeighborsClassifier(n_neighbors=1,
                            p=2,
                            metric='minkowski')

pipe1 = Pipeline([['sc', StandardScaler()],
                  ['clf', clf1]])
# tree-based classifiers don't have to standscaler operation!
pipe3 = Pipeline([['sc', StandardScaler()],
                  ['clf', clf3]])

# Majority Rule(hard) Voting by self-implementing class "MajorityVoteClassifier"
mv_clf1 = MajorityVoteClassifier(classifiers=[pipe1, clf2, pipe3])
mv_clf1.fit(X_train, y_train)
print("MajorityVoteClassifier(hard): ", mv_clf1.predict(X_test[:30]))

# Majority Rule(hard) Voting by SKLearn class "VotingClassifier"
mv_clf2 = VotingClassifier(estimators=[('lr',pipe1),('dt',clf2),('knn', pipe3)], voting='hard')
mv_clf2.fit(X_train, y_train)
print("VoteClassifier(hard)        : ", mv_clf2.predict(X_test[:30]))

# Majority Rule(soft) Voting by SKLearn class "VotingClassifier"
mv_clf3 = VotingClassifier(estimators=[('lr',pipe1),('dt',clf2),('knn', pipe3)], voting='soft')
mv_clf3.fit(X_train, y_train)
print("VoteClassifier(soft)        : ", mv_clf3.predict(X_test[:30]))
```

```
MajorityVoteClassifier(hard):  [0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 1.
 0. 1. 1. 0. 0. 0.]
VoteClassifier(hard)        :  [0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 1.
 0. 1. 1. 0. 0. 0.]
VoteClassifier(soft)        :  [0. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 0. 0. 1. 1.
 0. 1. 1. 0. 0. 0.]
```

```
clf_labels = ['Logistic regression', 'Decision tree', 'KNN', 'MajorityVote']

all_clf = [pipe1, clf2, pipe3, mv_clf1]

for clf, label in zip(all_clf, clf_labels):
    scores = cross_val_score(estimator=clf,
                             X=X_train,
                             y=y_train,
                             cv=10,
                             scoring='roc_auc')
    print("ROC AUC: %0.2f (+/- %0.2f) [%s]"
          % (scores.mean(), scores.std(), label))
```

```
ROC AUC: 0.91 (+/- 0.03) [Logistic regression]
ROC AUC: 0.99 (+/- 0.01) [Decision tree]
ROC AUC: 0.92 (+/- 0.03) [KNN]
ROC AUC: 1.00 (+/- 0.00) [MajorityVote]
```

From the figure of the previous page, we can clearly see that the result of the MajorityVoteClassifier is better than all the other models, even the roc_auc value is as high as 1, it is a prefect result. Therefore, we can say that training on an ensemble of numerous models and predicting an output (class) based on their highest probability of chosen class as the output is a helpful strategy. It seem to make sure to include a variety of models to feed a Voting Classifier to be sure that the error made by one might be resolved by the other.

Next, I performed the BaggingClassifier, It is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

```
bag = BaggingClassifier(base_estimator=tree,
                        n_estimators=500,
                        max_samples=1.0,
                        max_features=1.0,
                        bootstrap=True,
                        bootstrap_features=False,
                        n_jobs=1,
                        random_state=1)
```

```
ROC AUC: 1.00 (+/- 0.00) [Decision tree]
0.9954545454545455
ROC AUC: 1.00 (+/- 0.00) [Bagging]
1.0
ROC AUC: 1.00 (+/- 0.00) [Random Forest]
0.9997552447552448
```

I listed the roc_auc values of three models, for the accuracy, decision tree, BaggingClassifier and random forest can achieve to the perfect classifications, but I found that the result of BaggingClassifier is a little bit better than the other methods for the roc_auc values. Perhaps I can say that the advantages of BaggingClassifier still exist.

- Discuss possible problems you plan to investigate for future studies.

After writing this assignment, I can observe that the performance of the baseline model for the first dataset (adult dataset) are seem to be limited. I don't know why, but I think this is a problem I must try to conquer, because this problem may also appear in my final report.

The other problem is the comparison between MayjorityVoteClassifier and BaggingClassifier I want to do, because they have the similar performance on the second dataset.