Machine Learning Project: Classification and Creating Segments

https://github.com/yiic

Problem Description

- The aim is to build models to determine the income level of the people in U.S. It is a binary classification problem to predict if an individual has an income higher than $50k/year.
- Creating a segmentation model to predict which segment a new observation will belong to.

Data Exploration

Census-income data: this data set contains weighted census data extracted from 1994 and 1995
Check data

- Check Data shape
- Observe data
- Observation Summary
  The first thing I did was investigate the general shape of our data. feature_analysis.py kept track of the number of distinct levels of each feature written in file features_histogram.txt. My findings are as follows.
  - Some features contain '?'.
  - Some features have a continuous data, such as AAGE.
  - Majority dominants of some features have only two categories, such as AHRSPAY', 'CAPGAIN', 'CAPLOSS','DIVVAL'
  - This data set has an imbalanced problem, which means unevenly dataset of both positive and negative classes. (Below 50K 93.8%, Above 50K 6.2%)

Pre-processing approach

preprocess.py demonstrates the steps of pre-processing.

Steps

- replace '?' with NaN and set label value (1 if income == ' 50000+.' else 0)
- Combine AAGE to three categories (0-30, 31-60, 61 -90)
- Categorize 'AHRSPAY', 'CAPGAIN', 'CAPLOSS','DIVVAL' with Zero and MoreThanZero
- encode categorical data
- Balance data by Synthetic Data Generation(SMOTE)

Classification model

With cleaned data, we are ready to train our model. I ran three different algorithms: boosting, random forest, and logistic regression. Two methods, K-fold Cross Validation and Bootstrap, are used to evaluate these algorithms. Bootstrap also can train a model to predict new observations.

Steps:

- Three algorithms, logistic regression, random forest and boosting, are fit into K-fold Cross Validation and Bootstrap
- Fit a classifier algorithm to the cleaned data and assign instance weight provided by MARSUPWT.
- In the Bootstrp, I split the dataset into training and testing sets. 100 loops are set to train the model.
- Show Area Under the Curve(AUC) and Accuracy as a result

Results:

- Kfold cross validation
  - Logistic regression: AUC_mean: 0.9257, Accuracy_mean: 0.8573
  - Random forest: AUC_mean: 0.9913, Accuracy_mean: 0.9609
  - Boosting: AUC_mean: 0.9917, Accuracy_mean: 0.9534
- Bootstrap
  - Logistic regression: AUC_mean: 0.7178, Accuracy_mean: 0.7187
  - Random forest: AUC_mean: 0.8807, Accuracy_mean: 0.8815
  - Boosting: AUC_mean: 0.8687, Accuracy_mean: 0.8689

In this implementation, Random forest and Boosting achieve a better result than logistic regression. Besides, Kfold cross validation shows a better AUC and Accuracy than Bootstrap. In Bootstrap, we take 100 items as a sample in each iteration. The dataset is too small to get a good result. If I use a larger number of data as a sample, the performance of my computer is too slow to get result. In the future, I can try a larger sample on a powerful machine.

By running training code with Bootstrap method, we can get a trained model file. We'll load it into predict_classify.py and predict the target of a new observation.

Feature Selection

Not all of features determine the result. Sometimes, feature subsets give better results than complete set of features for the same algorithm.

- Advantages of feature selection
  - Train faster
  - Reduce the complexity of a model
  - Reduce overfitting
- Three method are selected and results are show in below table 1.
  - Forward feature selection
  - Random forest
  - Boosting

Table 1 Feature Selection

Forward feature selection	Random forest	Boosting
WKSWORK	DIVVAL	ACLSWKR
DIVVAL	ADTOCC	AHGA
ASEX	ASEX	AAGE
AAGE	AAGE	SEOTR
CAPGAIN	NOEMP	CAPGAIN
CAPLOSS	ADTIND	ASEX
AHRSPAY	AHGA	CAPLOSS
AREORGN	FILESTAT	AWKSTAT
NOEMP	AMJOCC	AUNMEM

- According to above table, a better set of features obtained: ['AAGE', 'AHGA', 'ASEX', 'CAPGAIN', 'CAPLOSS','DIVVAL', 'NOEMP','WKSWORK']. The paper [1] shows that "ASEX, AHGA, AAGE" are significant features to decide who earn more or less.
  Note: features of "WKSWORK, ADTOCC, ACLSWKR" only show once, but they are important features. So, I randomly select WKSWORK to feature subset.

Segmentation Model

Preprocess data

With cleaned data, we are ready to train our model. The dataset is composed of eights important features: ['AAGE', 'AHGA', 'ASEX', 'CAPGAIN', 'CAPLOSS','DIVVAL', 'NOEMP','WKSWORK'].

First, I consider a thing if one (or more) of the eight features is actually relevant. That is to say, is it possible to determine whether the people with a higher income have one feature will have another feature? I can make a determination by training a supervised regression learner on a subset of the data with one feature removed, and then score how well that model can predict the removed feature.

- Remove income feature as label and split the dataset into training and testing sets. (test: 25%, train:75%)
- Import a decision tree regressor and fit the learner to the training data.
- Report the prediction score

R2 score for AAGE as dependent variable: 0.25669149796058177
R2 score for AHGA as dependent variable: -0.03563212596492326
R2 score for ASEX as dependent variable: -0.0741525918631778
R2 score for CAPGAIN as dependent variable: -0.1256551646966395
R2 score for CAPLOSS as dependent variable: -0.33109120222640587
R2 score for DIVVAL as dependent variable: -0.01049348065993505
R2 score for NOEMP as dependent variable: 0.8188364820211318
R2 score for WKSWORK as dependent variable: 0.9249065316704972

The prediction score will imply the model is good or not to fit the data. (R2_score in train_seg.py)

Figure 1 shows a correlation matrix. The higher the color is on the bar, the higher the correlation. From the Figure 1, it can be found NOEMP and WKSWORK has a strongest correlation. How to dismiss the correlation will be covered in the future work.

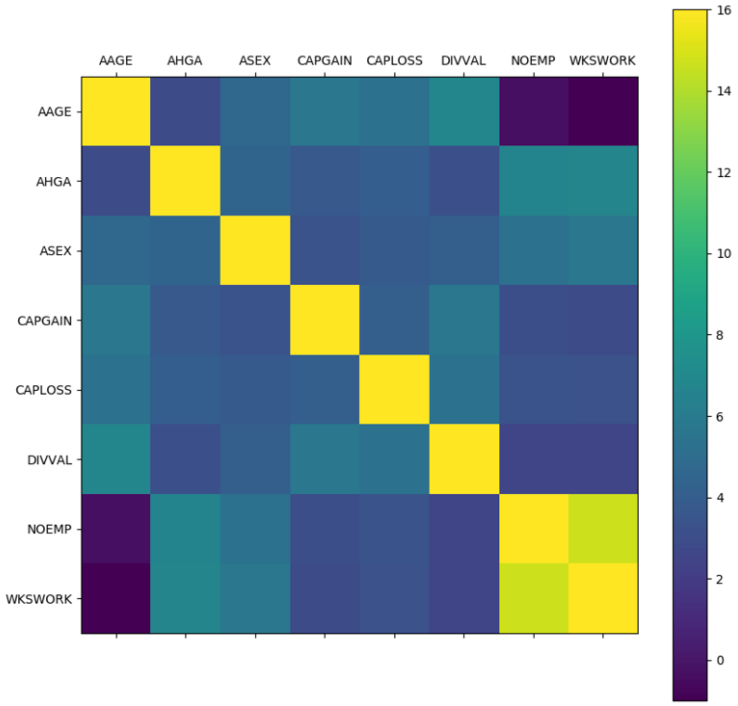

Figure 1 Features Correlation

Feature Transformation

In this section, I will use principal component analysis (PCA) to calculates the dimensions which best maximize variance, we will find which compound combinations of features best describe customers.
We got a result as shown in Figure 2. 92.80% of the variance explained by the first and second principal components.

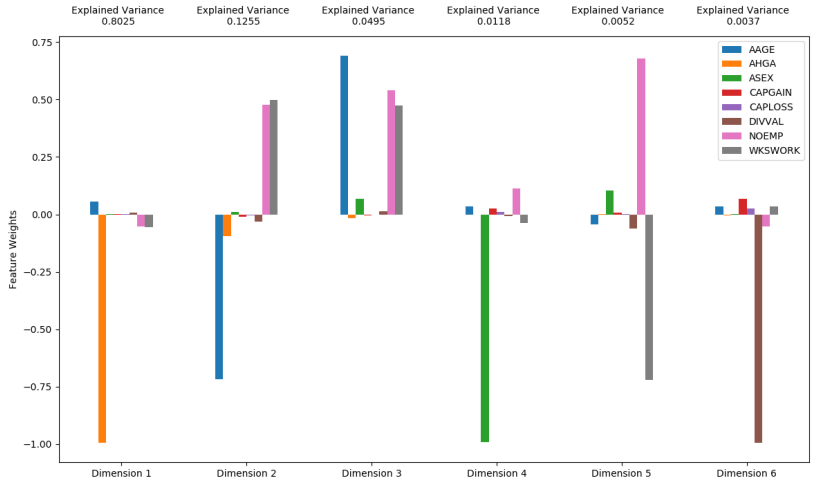

Figure 2 PCA

Clustering

In this section, I choose to use either a K-Means clustering algorithm or a Gaussian Mixture Model clustering algorithm to identify the various segments.

Depending on the problem, the number of clusters that you expect to be in the data may already be known. You can set the number of clusters by –n when running train_seg.py code.

Steps:

- Fit a clustering algorithm to the transformed data by PCA and assign it to clusterer.
- Predict the cluster for each data point in transformed data using clusterer.predict and assign them to preds.
- Find the cluster centers using the algorithm's respective attribute and assign them to centers.
- Predict the cluster for each sample data point in samples and assign them sample_preds.
- Calculate the silhouette score of transformed data against preds.
- Save cluster model to file.

Results:

KNN : Number of clusters is 10. The average silhouette score is: 0.645485798528627.
GMM : Number of clusters is 10. The average silhouette score is: 0.5529613515701454

From the result, KNN shows a better result than GMM. The Silhouette Coefficient is calculated using the mean intra-cluster distance and the mean nearest-cluster distance for each sample. Therefore, it makes sense to use the same distance metric here as the one used in the clustering algorithm.

Gaussian Mixture Models using information-theoretic criteria (BIC) could sometimes be a better criterion for deciding on the optimal number of clusters.

By running training code, we can get a trained model file. We'll load it into predict_seg.py and predict which segment a new observation will belong to.

Future Work

(1) I'll keep working on the method which dismiss the correlation of features.
(2) Instance weight is used to generate a classification model, but still have no idea on generating segmentation model. My understanding is the probability of each instance is proportional to the "instance weight". In the segmentation model, I pay more attention to the important features and their correlations.
(3) In the process of generating classification model, when a larger amount of data is used as a sample, it is too slow to get result.
(4) Above problem occurs in the processing of generation segmentation model. Therefore, I only use 30000 rows as training data.

Reference

[1] "More Education Means Higher Career Earnings", https://www.census.gov/topics/income-poverty/income/library/publications.1994.html
[2] https://elitedatascience.com/imbalanced-classes
[3] https://beckernick.github.io/oversampling-modeling/
[4] https://en.wikipedia.org/wiki/Market_segmentation
[5] https://github.com/ritchieng/machine-learning-nanodegree