

CS405 Machine Learning

Lab #2 Preliminary

Objectives : This lab will introduce how to pre-process and transform data to make machine-learning algorithm work. In this lab, you will employ several supervised algorithms of your choice to accurately model individuals' income using data collected from the 1994 U.S census. Your goal with this lab is to construct a model that accurately predicts whether an individual makes more than \$50000.

Exploring the Data

Run the code cell below to load necessary Python libraries and load the census data. Note that the last column from this dataset “income”, will be our target label (whether an individual makes more than, or at most, \$50,000 annually). All other columns are features about each individual in the census database.

```
[1]: # Import libraries necessary for this project
import numpy as np
import pandas as pd
from time import time
from IPython.display import display # Allows the use of display() for DataFrames

# Import supplementary visualization code visuals.py
import visuals as vs

# Pretty display for notebooks
%matplotlib inline

# Load the Census dataset
data = pd.read_csv("census.csv")

# Success - Display the first record
display(data.head(n=1))
```

	age	workclass	education_level	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	Bachelors	13.0	Never-married	Adm-clerical	Not-in-family	White	Male	2174.0	0.0	40.0	United-States	<=50K

Exercise 0:

A cursory investigation of the dataset will determine how many individuals fit into either group, and will tell us about the percentage of

these individuals making more than \$50,000. In the code cell below, you will need to compute the following:

- The total number of records, 'n_records';
- The number of individuals making more than \$50000 annually, 'n_greater_50k'.
- The number of individuals making at most \$50000 annually, 'n_at_most_50K'.
- The percentage of individuals making at more than \$50000 annually, 'greater_percent'
- Feature values for each column

Tips :As the data is stored as pandas, this tutorial will help you finish <https://pandas.pydata.org/pandas-docs/stable/10min.html>

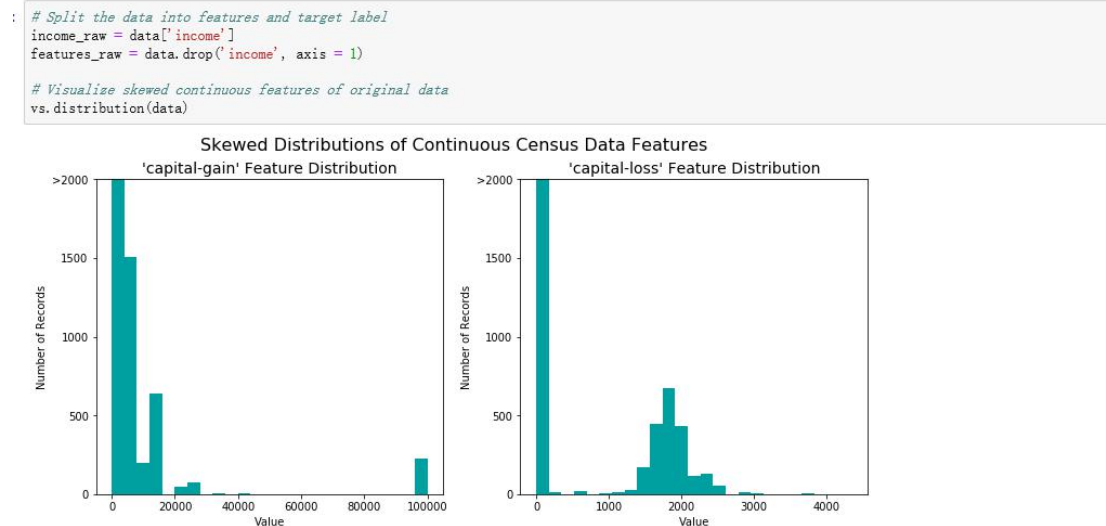
Preparing the Data

Before the data can be used as the input for machine learning algorithms, it often must be cleaned, formatted, and restructured — this is typically known as preprocessing. Fortunately, for this dataset, there are no invalid or missing entries we must deal with, however there are some qualities about certain features that must be adjusted. This preprocessing can help tremendously with the outcome and predictive power of nearly all learning algorithms.

Transforming Skewed Continuous Features

A dataset may sometimes contain at least one feature whose values tend to lie near a single number, but will also have a non-trivial number of vastly larger or smaller values than that single number. Algorithms can be sensitive to such distributions of values and can underperform if the range is not properly normalized. With the census dataset two features fit this

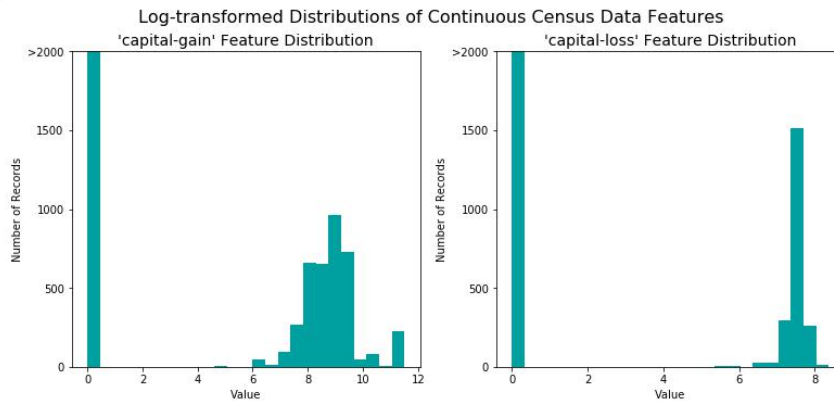
description: 'capital-gain' and 'capital-loss'. The code cell below will plot a histogram of these two features. Note the range of the values present and how they are distributed.



For highly-skewed feature distributions such as 'capital-gain' and 'capital-loss', it is common practice to apply a logarithmic transformation on the data so that the very large and very small values do not negatively affect the performance of a learning algorithm. Using a logarithmic transformation significantly reduces the range of values caused by outliers. Care must be taken when applying this transformation however: The logarithm of 0 is undefined, so we must translate the values by a small amount above 0 to apply the the logarithm successfully. Below code cell will perform a transformation on the data and visualize the results. Again, note the range of values and how they are distributed.

```
8]: # Log-transform the skewed features
skewed = ['capital-gain', 'capital-loss']
features_log_transformed = pd.DataFrame(data = features_raw)
features_log_transformed[skewed] = features_raw[skewed].apply(lambda x: np.log(x + 1))

# Visualize the new log distributions
vs.distribution(features_log_transformed, transformed = True)
```



Normalizing Numerical Features

In addition to performing transformations on features that are highly skewed, it is often good practice to perform some type of scaling on numerical features. Applying a scaling to the data does not change the shape of each feature's distribution (such as 'capital-gain' or 'capital-loss' above); however, normalization ensures that each feature is treated equally when applying supervised learners. Note that once scaling is applied, observing the data in its raw form will no longer have the same original meaning, as exemplified below.

```
: # Import sklearn.preprocessing.StandardScaler
from sklearn.preprocessing import MinMaxScaler

# Initialize a scaler, then apply it to the features
scaler = MinMaxScaler() # default=(0, 1)
numerical = ['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']

features_log_minmax_transform = pd.DataFrame(data = features_log_transformed)
features_log_minmax_transform[numerical] = scaler.fit_transform(features_log_transformed[numerical])

# Show an example of a record with scaling applied
display(features_log_minmax_transform.head(n = 5))
```

	age	workclass	education_level	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country
0	0.301370	State-gov	Bachelors	0.800000	Never-married	Adm-clerical	Not-in-family	White	Male	0.667492	0.0	0.397959	United-States
1	0.452055	Self-emp-not-inc	Bachelors	0.800000	Married-civ-spouse	Exec-managerial	Husband	White	Male	0.000000	0.0	0.122449	United-States
2	0.287671	Private	HS-grad	0.533333	Divorced	Handlers-cleaners	Not-in-family	White	Male	0.000000	0.0	0.397959	United-States
3	0.493151	Private	11th	0.400000	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0.000000	0.0	0.397959	United-States
4	0.150685	Private	Bachelors	0.800000	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0.000000	0.0	0.397959	Cuba

Data Preprocessing

From the table in above, we can see there are several features for each record that are non-numeric. Typically, learning algorithms expect input to be numeric, which requires that non-numeric features (called ‘categorical variables’) be converted. One popular way to convert categorical variables is by using the one-hot encoding scheme. One-hot encoding creates a ‘dummy’ variable for each possible category of each non-numeric feature. For example, assume some features has three possible entries: A, B and C. We then encode this feature into someFeature_A, someFeature_B and someFeature_C.

someFeature		someFeature_A	someFeature_B	someFeature_C
0	B	0	1	0
1	C	0	0	1
2	A	1	0	0

Additionally, as with the non-numeric features, we need to convert the non-numeric target label, ‘income’ to numerical values for the learning algorithm to work. Since there are only two possible categories for this label (“<=50K” and “>50K”), we can avoid using one-hot encoding and simply encode these two categories as 0 and 1, respectively.

age	education-num	capital-gain	capital-loss	hours-per-week	workclass_Federal-gov	workclass_Local-gov	workclass_Private	workclass_Self-emp-inc	workclass_Self-emp-not-inc	...	native-country_Portugal	native-country_Puerto-Rico	native-country_Scotland	native-country_South	native-country_Taiwan	native-country_Thailand
0.301370	0.800000	0.667492	0.0	0.397959	0	0	0	0	0	...	0	0	0	0	0	0
0.452055	0.800000	0.000000	0.0	0.122449	0	0	0	0	1	...	0	0	0	0	0	0
0.287671	0.533333	0.000000	0.0	0.397959	0	0	1	0	0	...	0	0	0	0	0	0
0.493151	0.400000	0.000000	0.0	0.397959	0	0	1	0	0	...	0	0	0	0	0	0
0.150685	0.800000	0.000000	0.0	0.397959	0	0	1	0	0	...	0	0	0	0	0	0

Exercise 1:

- Perform one-hot encoding on the data
- Convert the target label ‘income_raw’ to numerical entries (set records with “<=50k” to 0 and records with “>50k” to 1).

Shuffle and Split Data

Tips: pandas.get_dummies() can perform one-hot encoding.

When all categorical variables have been converted into numerical features, and all numerical features have been normalized. As always, we will now split the data (both features and their labels) into training and test sets. 80% of the data will be used for training and 20% for testing.

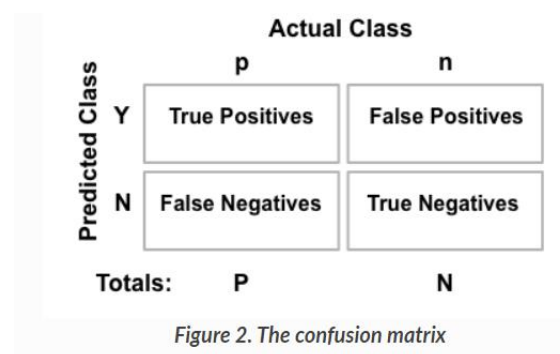
```
# Import train_test_split
from sklearn.model_selection import train_test_split

# Split the 'feature' and 'income' data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_final, income, test_size = 0.2, random_state = 0)

# Show the results of the split
print("Training set has {} samples.".format(X_train.shape[0]))
print("Testing set has {} samples.".format(X_test.shape[0]))
```

Training set has 36177 samples.
Testing set has 9045 samples.

Evaluating Model Performance



● Accuracy

Accuracy measures how often the classifier makes the correct prediction. It's the ratio of the number of correct predictions to the total number of predictions.

$$ACC = \frac{TP + TN}{TP + FP + TN + FN}$$

In Python you can calculate it in the following way:

```
from sklearn.metrics import confusion_matrix, accuracy_score

y_pred_class = y_pred_pos > threshold
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_class).ravel()
accuracy = (tp + tn) / (tp + fp + fn + tn)
# or simply
accuracy_score(y_true, y_pred_class)
```

● Precision

Precision tells us what proportion of messages we classified as positive.

It is a ratio of true positives to all positive predictions. In other words,

$$\text{Precision} = \text{TP} / (\text{TP} + \text{FP})$$

● Recall

Recall(sensitivity) tells us what proportion of messages that actually were positive were classified by us as positive.

$$\text{Recall} = \text{TP} / (\text{TP} + \text{FN})$$

● F1 score

We can use **F-beta score** as a metric that considers both precision and

$$\text{recall: } F_{\beta} = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

When choosing beta in your F-beta score **the more you care about recall** over precision **the higher beta** you should choose. For example, with **F1 score we care equally about recall and precision** with F2 score, recall is twice as important to us.

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html#sklearn.metrics.f1_score

● TPR & FPR & ROC & AUC

$$\text{TPR} = \frac{\text{positives}_{\text{correctly_classified}}}{\text{total}_{\text{positive}}} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{P}$$

$$\text{FPR} = \frac{\text{negatives}_{\text{incorrectly_classified}}}{\text{total}_{\text{negatives}}} = \frac{\text{FP}}{\text{TN} + \text{FP}} = \frac{\text{FP}}{N}$$

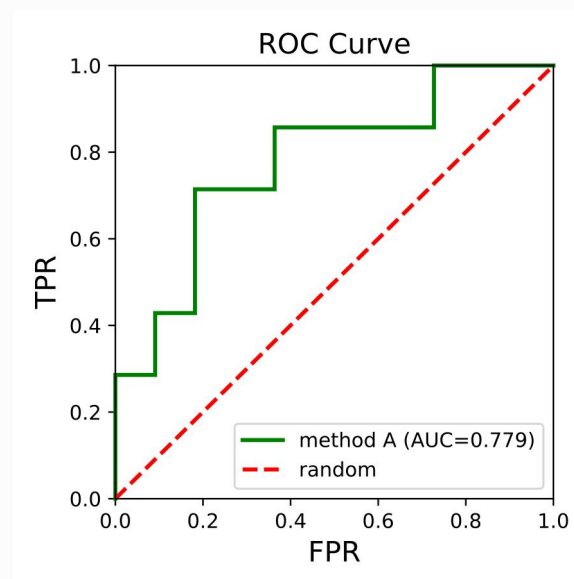
ROC (Receiver Operating Characteristic) is used to measure the output quality of the evaluation classifier.

ROC curves are two-dimensional graphs in which true positive rate (TPR) is plotted on the Y axis and false positive rate (FPR) is plotted on the X axis. An ROC graph depicts relative tradeoffs between true positive rate

(TPR) and false positive rate (FPR). Basically, for every threshold, we calculate TPR and FPR and plot it on one chart.

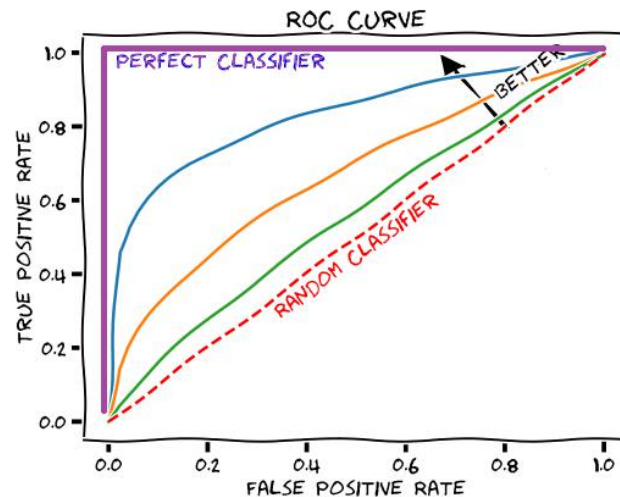
Example data for ROC:

ID	Actual	Prediction Probability	>0.6	>0.7	>0.8	Metric
1	0	0.98	1	1	1	
2	1	0.67	1	0	0	
3	1	0.58	0	0	0	
4	0	0.78	1	1	0	
5	1	0.85	1	1	1	
6	0	0.86	1	1	1	
7	0	0.79	1	1	0	
8	0	0.89	1	1	1	
9	1	0.82	1	1	1	
10	0	0.86	1	1	1	
			0.75	0.5	0.5	TPR
			1	1	0.66	FPR
			0	0	0.33	TNR
			0.25	0.5	0.5	FNR



Example of ROC curve

The higher TPR and the lower FPR is for each threshold the better and so classifiers that have curves that are more top-left-side are better.



AUC (Area Under Curve) means area under the curve, it is a performance metric that you can use to evaluate classification models. There are functions for calculating AUC available in many programming languages. In python , you can refer following link :

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html#sklearn.metrics.roc_auc_score

Exercise 2: Now we chose a model that always predicted an individual made more than \$50,000, what would that model's accuracy and F-score be on this dataset? You must use the code cell below and assign your results to 'accuracy' and 'f-score' to be used later.

Exercise 3: The following are some of the supervised learning models that are currently available in *scikit-learn* :

- Gaussian Naive Bayes (GaussianNB)
- Decision Trees
- Ensemble Methods (Bagging, AdaBoost, RandomForest)
- K-Nearest Neighbors
- Support Vector Machines (SVM)
- Logistic Regression

You choose three of them, draw three ROC curves on the census data, and analyze and compare the them.

Questions:

- (1) An important task when performing supervised learning on a dataset like the census data we study here is determining which features provides the most predictive power. Choose a *scikit-learn* classifier (e.g. adaboost, random forests) that has a `feature_importance_` attribute, which is a function that ranks the importance of features according to the chosen classifier. List three of the supervised learning models above that are appropriate for this problem that you will test on the census data.
- (2) Describe one real-world application in industry where the model can be applied
- (3) What are the strengths of the model; when does it perform well?
- (4) What are the weaknesses of the model; when does it perform poorly?
- (5) What makes this model a good candidate for the problem, given what you know about the data?