

Programming Project: What Makes People Happy

Instructor: Prof. John C.S. Lui

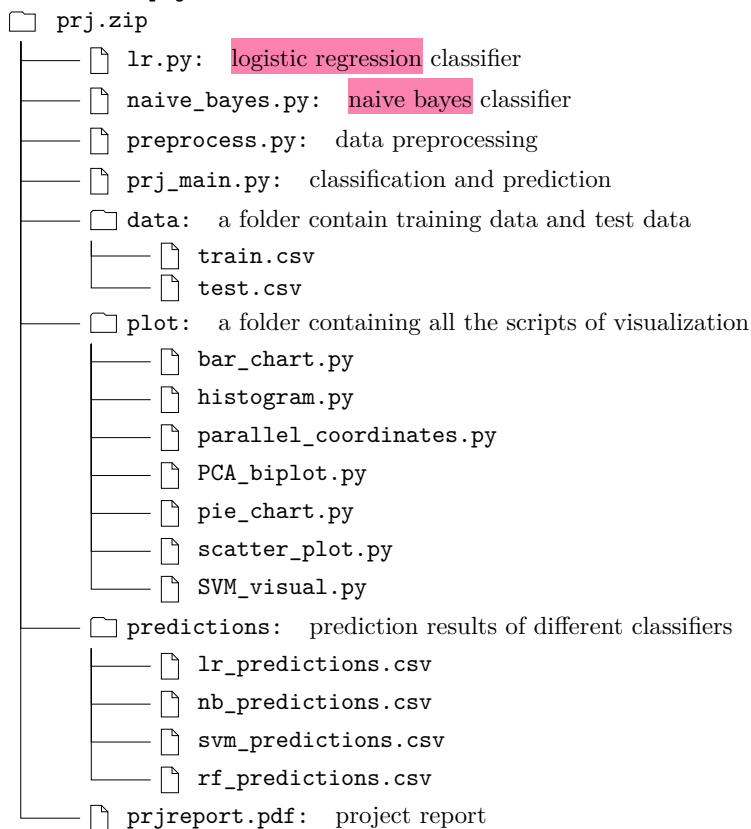
Due: 23:59 on Tuesday, May. 02, 2017

1 Introduction

What predicts happiness? In this project, you'll use data from [Show of Hands](#), an informal polling platform for use on mobile devices and the web, to see what aspects and characteristics of people's lives predict happiness. Show of Hands has been downloaded over 300,000 times across Apple and Android app stores, and users have cast more than 75 million votes. In this problem, we'll use data from thousands of users and one hundred different questions to see which responses predict happiness. You can also this problem as a competition in the data science competition platform [Kaggle](#).

1.1 File Descriptions





To start, you need to download the **prj.zip** file from the course website. Unzip it and you will find the structure of the prj folder as follows:



1.2 Requirements

To complete this project, you need to follow the instructions carefully. Specifically, it should be noted that:

1. For this programming project, please use Python 3.

2. This project is a group programming project. Two students form a group. Each group should submit their code and report.
3. Meaning of . When you meet an symbol , it means that you need to write or add some Python scripts in a corresponding *.py script file.
4. Meaning of . When you meet a  symbol, it means that you need to write down your answers to the questions marked with **Q** to the project report file `prjreport.pdf`.

2 Raw Data Preprocessing

Machine learning algorithms learn from data. However, the data we can actually obtain is not born for learning. Instead, the raw data is usually incorrect, incomplete, improperly formatted, duplicated, or in some even worse cases. It is critical to feed those learning algorithms the right data for the problem we want to solve. Even if we have good data, we need to make sure that it is in a useful scale, format and even that meaningful features are included. Hence, data scrubbing, also called **data cleansing**, is a very important step to prepare data for a machine learning algorithm. And it generally include the following steps:

1. Data analysis
2. **Missing data** processing
3. **Outlier** processing
4. **Redundant** data processing
5. **Noisy** data processing

Now, let's focus on the data preprocessing of this project, we mainly focus on **"data analysis"** and **"missing data processing"**. These two steps are required in this project. More detailed requirements of relevant tasks will be specified in the subsequent parts. However, to pursue better classification and inference performance, you are also encouraged to trying other steps.

2.1 Get Start With Pandas

For data pre-processing, **pandas** is a great Python package. To represent tabular data, pandas uses a custom data structure called a dataframe. A dataframe is a highly efficient, **2-dimensional** data structure that provides a suite of methods and attributes to quickly explore, analyze, and visualize data. The dataframe is similar to the **NumPy 2D** array but adds support for many features that help you work with tabular data. To learn how to explore the raw data with **pandas**, please read the document of `read_csv()` in [pandas official website](#). After loading `train.csv` with pandas, the data we obtained is in the following shape.

UserID	YOB	Gender	Income	HouseholdStatus	EducationLevel	Party	Q124742	...
1	1938	Male	nan	Married (w/kids)	nan	Independent	No	...
2	1985	Female	\$25,001 -\$50,000	Single (no kids)	Master's Degree	Democrat	nan	...
5	1963	Male	over \$150,000	Married (w/kids)	nan	nan	No	...
...
9480	nan	Female	nan	nan	nan	Independent	nan	...
9503	1945	Male	\$25,001 -\$50,000	Married (w/kids)	High School Diploma	Democrat	nan	...

Now that we've read the dataset into a dataframe, we can start using the dataframe methods to explore the data. The Series object is a core data structure that pandas uses to represent rows and columns. A Series is a labelled collection of values similar to the NumPy vector. But instead of utilizing integer labels for indexing, Pandas is able to index with features as **non-interger labels**. For example, when you select a row

from a dataframe with `loc[]`, instead of just returning the values in that row as a list, pandas returns a Series object that contains the column labels as well as the corresponding values:

UserID	YOB	Gender	Income	HouseholdStatus	EducationLevel	Party	Q124742	...
1	1938	Male	nan	Married (w/kids)	nan	Independent	No	...

Pandas can detect data type, we can use the `df.dtypes` command to check the data type read with Pandas. Here, `dtype:` object refers to the data type, or dtype, of that Series. The object dtype is equivalent to the string type in Python.

UserID	YOB	Gender	Income	HouseholdStatus	EducationLevel	Party	Q124742	...
in64	float64	object	object	object	object	object	object	...

Pandas also provides many statistic functions, such as `mean()`, `std`. We can also use `df.info()` to show us some basic knowledge of raw data, and `df.describe()` to give us an overview of the raw data.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4619 entries, 0 to 4618
Columns: 110 entries, UserID to votes
dtypes: float64(1), int64(3), object(106)
memory usage: 3.9+ MB
```

```

      UserID      YOB      Happy      votes
count  4619.000000  3935.000000  4619.000000  4619.000000
mean    3830.143538  1979.135197    0.563758    71.911669
std     2375.217598   15.316854    0.495972    28.549388
min         1.000000  1900.000000    0.000000    20.000000
25%    1769.500000         NaN    0.000000    45.000000
50%    3717.000000         NaN    1.000000    82.000000
75%    5674.000000         NaN    1.000000    99.000000
max    9503.000000  2039.000000    1.000000   101.000000
[[1 1938.0 0.0 ..., 13 24 86]
 [2 1985.0 1.0 ..., nan nan 66]
 [5 1963.0 0.0 ..., 24 24 101]
 ...,
 [9480 1982.0 1.0 ..., nan nan 20]
 [9503 1945.0 0.0 ..., nan nan 20]]
```

2.2 Data Discretization and Analysis

After analyzing the raw data with Pandas, we know that there are 106 `object` type attributes of the training samples. The `float64` type attribute `YOB` and the `int64` type attributes `UserID`, `Happy`, `votes` are already discrete numerical values. But we still need to map the left 106 `object` type attributes to discrete numerical values. To do this, you can use the `map()` command or assign new values for different columns with `loc[]` command. You are required to finish this step by implementing the function `transform` in `preprocess.py`. After this step, all `object` type attributes should be transformed to discrete numerical values, for example:

	UserID	YOB	Gender	Income	HouseholdStatus	EducationLevel	Party	...
0	1	1938.0	0	nan	3	nan	0	...
1	2	1985.0	1	1	0	5	1	...
2	5	1963.0	0	5	3	nan	nan	...
...								
4617	9480	nan	1	nan	nan	nan	0	...
4618	9503	1945.0	0	1	3	1	1	...

And your code in function `transform` should in the following format.

```
def transform(filename):  
    # your code here  
    return {'data':data,'target':target}
```

You should return a `dict` data structure with `'data'` and `'target'` as the keys. And the value corresponding to `'data'` is a $n \times m$ 2d array where n is the number of samples and m is the number of features; the value corresponding to `'target'` is a 1d array with n entries recording the `Happy` information of samples.

We can use the statistic functions of Pandas to help us understand the quality of different attributions and help us to select some useful attributes as features. You are encouraged to try different subset of attributes as features. Feel free to put down your `analysis` and `observations`. The tasks of this subsection are summarized as follows:

- ⇒ Complete the `discretization` of raw data in function `transform`. The interfaces should comply with the requirement above. An `optional` task is to do statistic analysis of data and try different subset of attributes as features.
- ✍ An `optional` task is to do `statistic analysis of data` and try different subset of attributes as features. Feel free to put down your analysis and observations in `prjreport.pdf`.

2.3 Missing Data Filling

You may notice that the obtained data in `transform` still contains many missing values, and now we need to handle them.

There are two types of missing data, which need to be handled in different ways. One possibility is that there are reasons why the data is missing. In this case, whether some part of the data is missing carries important information and could be used for machine learning algorithms. The other possibility is that there are no reasons why the data is missing, and whether some part of the data is missing carries no information.

Let us first consider the first possibility where missing data is informative. When we believe that there is information in the data being missing, we usually want the machine learning algorithm to be able to use this information to `potentially improve the prediction accuracy`. For numerical attribute, we simply pick a number in one end of the numerical spectrum that will `denote` missing values, such that it will not be in the middle of the distribution of non-missing values. For category attribute, the missing value becomes their own category.

Now let us deal with the other scenarios where missing data is `not informative`. We need to handle the missing data in different ways. More specifically, we can:

- `Drop the missing data`. This is the most straightforward way. If we have sufficient samples and attributes, and the missing data is very sparse and appears randomly, we take this method.
- `Replace missing data` with some statistic values, e.g., mean, median or mode. The advantage of this method is it is simple and statistic sufficient, i.e., this replacement will not reduce the information carried by those samples. Generally speaking, if the missing data follows some distribution, we can use mean; otherwise we can use median.
- `Use interpolation method`. We can do randomly interpolation by replace the missing data with a value randomly selected from samples, or use numerical interpolations such as Lagrange's polynomial interpolation and Newton's interpolation. The numerical interpolations are simple and efficient.
- `Use modeling method`. We can use regression, Bayes and decision tree methods to help us infer the missing data. For example, we can use available samples to build `a decision tree and infer the missing data`.

To let you know in advance, we have two tasks in this part:

- In the basic task, we need to implement the method to replace missing data directly with `statistic values`, including `mean, median and mode`.

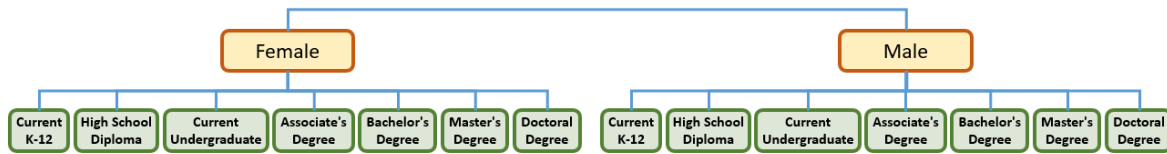


Figure 1: Example of classification tree

- For the advanced task, we firstly separate users into different groups and then use statistic values of the corresponding groups to replace the missing data. For example, we group the user by gender and education level so as to infer the missing income value in each group. In the first step, we classify users into the fourteen groups as shown in Fig. 1.

After that, we calculate the statistic values including mean, median and mode for each group. In the last step, the missing values are replaced by the statistic values for the corresponded groups.

In summary, you need to finish the following missing data filling function `fill_missing` in `preprocess.py`. The function `fill_missing` has the following prototype.

```
def fill_missing(X, strategy, isClassified):
    # your code here
    return X_full
```

Here X is a $n \times m$ 2d array where n is the number of samples and m is the number of features. The `strategy` we can choose includes `'mean'`, `'median'` and `'mode'`. `isClassified` is a boolean value indicate whether we will consider to use classification tree. The function returns the matrix `X_full`, which is obtained by filling missing data in `X`. Please pay attention to the classification indicators you will use to infer different attributes. Think about how to select suitable classification indicators. Feel free to put down your reasons of classification indicators selection in `prjreport.pdf`.

The tasks of this subsection are summarized as follows:

- Implement the missing data filling method `fill_missing` in file `preprocess.py`.
- Explain your reasons for the way you group the users. Feel free to put down your analysis and observations in `prjreport.pdf`.

3 Classification

Now with the training data, you can train classifiers you have learned in this course. Here you are required to build four classifiers: logistic regression classifier, Naive Bayes classifier, SVM classifier and random forest classifier. Besides, you also need to implement two of the classifiers, i.e., logistic regression classifier and Naive Bayes classifier.

3.1 Train Classifiers in Scikit-Learn

3.1.1 Logistic Regression

Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. You can find the detailed introduction to logistic regression in the User Guide of scikit-learn [1].

- Add scripts to the function `main()` in the file `prj_main.py` to train a logistic regression model `lr_model` with the training data. Please name your logistic regression model with the exact name `lr_model`.

- Implement your own Logistic regression classifier. Complete the class `LogisticRegression` defined in the file `lr.py`. Do not change the prototype of the functions `fit` and `predict`. The following statements should be valid.

```
import LogisticRegression from lr
.....
clf = LogisticRegression()
clf = clf.fit(X, y)
y_predict = clf.predict(X)
```

- Compare your implementation with the corresponding classifier implemented in scikit-learn in running time and train accuracy. Report your comparison result in `prjreport.pdf`.

3.1.2 Naïve Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features. You can find the details of Naive Bayes method in the User Guide of scikit-learn [2]. To train a Naive Bayes classifier, you need to the following.

- Choose a Naive Bayes classifier from `GaussianNB`, `MultinomialNB`, and `BernoulliNB` for the problem, and state your reason to choose the specific classifier in the project report `prjreport.pdf`.
- Add scripts to the function `main()` in the file `prj_main.py` to train a selected Naive Bayes classifier `nb_model` with the training data. Please name your Naive Bayes classifier with the exact name `nb_model`.
- Implement the selected Naive Bayes classifier and complete the class `NaiveBayes` in the file `naive_bayes.py`. The following statements should be valid.

```
import NaiveBayes from naive_bayes
.....
clf = NaiveBayes()
clf = clf.fit(X, y)
y_predict = clf.predict(X)
```

- Compare your implementation with the corresponding classifier implemented in scikit-learn in running time and train accuracy. Report your comparison result in `prjreport.pdf`.

3.1.3 SVM

Support vector machines (SVMs) are typical classification methods. You can find the details of SVM in the User Guide of scikit-learn [3]. To train an SVM classification model, you are recommended to use the `SVC` classifier in the scikit-learn module `sklearn.svm`.

- `SVC` accepts different kernel functions. Kernel functions could be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable function defined by yourself. Please choose a kernel function and state your reasons in the `prjreport.pdf`.
- Add scripts to the function `main()` in the file `prj_main.py` to train an SVM classifier `svm_model` with the training data. Please name your SVM classifier with the exact name `svm_model`.

3.1.4 Random Forest

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. You can find the details of random forest in the User Guide of scikit-learn [4]. To train a random forest classifier, you are recommended to use the `RandomForestClassifier` classifier in the scikit-learn module `sklearn.ensemble`.

- ➡ Add scripts to the function `main()` in the file `prj_main.py` to train a random forest classification model `rf_model` with the training data. Please name your random forest classifier with the exact name `rf_model`.

3.1.5 Tips in Training Classification Models

Learning the parameters of a prediction function and testing it on the same data is a methodological mistake: a model that would just repeat the labels of the samples that it has just seen would have a perfect score but would fail to predict anything useful on yet-unseen data. This situation is so-called overfitting. To avoid overfitting, you are recommended to use the cross validation techniques in scikit-learn. Such techniques can also help you tune the parameters of your classifiers. For information about cross validation, you can read the documentation of scikit-learn¹. Generally, you can get more accurate models if you use the cross validation techniques to tune your classifiers. You will also get additional marks if you use cross validation techniques in training your classifiers.

3.2 Get Predictions

We provide 924 unlabeled sample in the file `test.csv`. You are required to obtain the predictions on the test data using your four trained classifiers.

- ➡ Add scripts to the file `prj_main.py` to get and save prediction results of different classification models (You need read the test data by yourself. Hint: Use similar method as used in obtaining the training data.). Please save the predictions of `lr_model` in the file `lr_predictions.csv`, the predictions of `nb_model` in the file `nb_predictions.csv`, the predictions of `svm_model` in the file `svm_predictions.csv` and the predictions of `rf_model` in the file `rf_predictions.csv`. Note all these prediction files should be in the folder `predictions`. The format of these `*_predictions.csv` files is described as follows:

```
UserID,Happy
2,1
7,0
8,1
...
9480,0
9503,0
```

In particular, the first line is `UserID,Happy` and the following each line is printed with the format string `%d,%d\n`. (Only a single comma, without any space is in the middle of the two `%d`.) Note that the comma, splits the line into two columns. Each number in the left column is the user id and each number in the right column the indicators of whether the people is happy (1) or not (0).

Your prediction results should be **reproducible**. That is, your prediction results should be the same each time when you run your scripts. Your marks would be deducted if we find your prediction results and the running results of your scripts are not consistent. Hint: set the parameter `random_state` to be a fixed value in all the functions that provide this parameter.

We will compare your results with our ground truth data. The more accurate of your prediction results, the higher score you will obtain.

3.3 Write A Report

After you obtain all the results, you are required to write a brief report in the file `prjreport.pdf` to answer the following questions. You can write freely as long as you answer the questions clearly since there's no strict format for the report.

- 📖 Q: What are the characteristics of each of the four classifiers?
- 📖 Q: Different classification models can be used in different scenarios. How do you choose classification models for different classification problems? Please provide some examples.

¹http://scikit-learn.org/stable/modules/cross_validation.html

📌 Q: How do the **cross validation** techniques help in avoiding overfitting?

4 Visualization

Plain data can't be well perceived by us human beings. The most effective way to represent data is to visualize it using figures. Visualization is helpful when we want to understand the data and when we use machine learning algorithms to do something with the data.

In this section, we will visualize the data and the machine learning algorithms in various forms using Python. We only use the training data in this part. A plotted figure should convey clear information, so we have some basic requirements in this section:

- All the plots should have titles indicating the key message of the figure. You could give your own title.
- All the plots should have label texts for all the axes.

4.1 Basic Visualization Methods

4.1.1 Histogram of YOB

Histogram² is a way to visualize the distribution of quantitative data. Here, we want to see the **distribution of YOB** in the **given dataset**.

- 📌 Write script in `plot/histogram.py` to plot the histogram of **YOB (year of birth)**. The x-axis is the **YOB**, and the y-axis is the **probability density**. *Hint*: You could use `hist()` in `matplotlib`.

📌 Copy down this figure to your report. Then use one sentence to describe the information you observe from the figure. *Hint*: You may save your plotted figure using `savefig()` in `matplotlib`.

4.1.2 Pie chart for the fraction of happy men/women

Pie chart³ is a way to visualize the distribution of categorical data. In this part, we explore the relationship between **gender** and **happiness**.

- 📌 Write script in `plot/pie_chart.py` to draw a pie chart of **happy** and **unhappy** populations for **men** and another pie chart of **happy** and **unhappy** populations for **women**. These two pie charts should be plotted in the same figure. The categories in the pie chart should be "happy" and "unhappy". And there should be **label text** for each category on the figure. *Hint*: You may use `subplot()` and `pie()` in `matplotlib`.

📌 Copy down this figure to your report. And describe what you observe about the relationship between gender and happiness in one sentence.

4.1.3 Scatter plot of YOB and income

Scatter plot is a common way to visualize the relationship between two quantitative variables. In the following, we investigate on the relationship between YOB and income. Looking forward to our future, we may wonder, is it true that **the older the richer?**

- 📌 Write script in `plot/scatter_plot.py` to plot the relationship between **YOB and income**. The x-axis is the YOB and the y-axis is the income. Since we only know the income range of a questioned individual, we use the mean value of this range as the income of the individual. For example, when the reported income is "\$50,000 - \$74,999", we regard the income as **\$62500**. *Hint*: You may use `scatter()` in `matplotlib`, and you could set the **alpha** blending value to make the dots more transparent in order to view some pattern between YOB and income.

📌 Copy down this figure to your report. And describe what you observe about the relationship between **YOB and income** in one sentence.

²<https://en.wikipedia.org/wiki/Histogram>

³https://en.wikipedia.org/wiki/Pie_chart

4.1.4 Bar chart of income and happiness

Bar chart is suitable to visualize the relationship between **two categorical variables**. How one's income will affect one's happiness? In this part, we will have some clues from our data.

- Write script in `plot/bar_chart.py` to illustrate the relationship between **income and happiness**. The x-axis is for different **categories of income levels**, and the y-axis is for the fraction of happy/unhappy people within a certain income level. For every income level, there should be two *adjacent* bars corresponding to the fraction of the happy and the fraction of the unhappy respectively. Legend texts to describe which bar is for the happy(or unhappy) are necessary. *Hint*: We could use `bar()` in `matplotlib`

- Copy down the above figure in your report. Then write down your observation for the relationship of income and happiness in one sentence.

4.2 Visualizing High-dimensional Data

As a matter of fact, it's almost impossible for our eyes to see things with more than 3 dimensions. But the data is usually high dimensional. Many methods are proposed to deal with this problem of visualizing high-dimensional data. The common idea is to represent these high-dimensional data in 2-D or 3-D spaces. In the case where the number of data dimensions is small, we may think of projecting the data to low dimensional space. In the case where the number of data dimensions is large, we turn to the algorithms to visualize the data. In the following, we use two examples to show how these methods work.

4.2.1 Parallel Coordinates Plot

Parallel coordinates are a common way of visualizing high-dimensional geometry and analyzing multivariate data.⁴ We use each parallel vertical axis for each dimension, and represent a data entry by a horizontal polyline crossing all the corresponded points on the vertical axes.

- Write script in `plot/parallel_coordinates.py` to plot the data **with respect to only 5 dimensions** of "Gender", "Income", "HouseholdStatus", "EducationLevel" and "Party" **only for the first 100 data entries** in the training data. We have two categories of data, i.e. "happy" and "unhappy". Use **red** polylines for data points with the label "happy", and use **blue** polylines for data points with label "unhappy". The axes for each dimensions should have text labels. In this figure, you could represent the categories by discrete numbers instead of label texts, as long as you have clear description in the document. For example, you could use 0 for "male" and 1 for "female". *Hint*: The package `pandas` has a function `parallel_coordinates()`.

- Copy down the above figure to your report.

4.2.2 PCA and biplot

PCA as a dimensionality deduction method is not only useful to reduce the computation complexity, but also is a weapon to visualize high dimensional data. Biplot⁵ is a generalization of two-variable scatterplot. It allows information on both samples and variables of a data matrix to be displayed graphically.

- Write script in `plot/PCA_biplot.py` to do Principal Componets Analysis on the dataset. And add script to draw the biplot to visualize PCA results. In the biplot, the x-axis is for the first principle component, namely PC1, and the y-axis is for the second principle component, namely PC2. A scatter plot of all data points is plotted projecting onto the space of PC1 and PC2. All the variables are plotted as vectors in the same figure. Note that all the categories variables are treated as discrete variables after preprocessing. *Hint*: There is no off-the-shelf python package specially designed for biplot, you may learn from this [example](#) and see whether it satisfies our requirements.

⁴https://en.wikipedia.org/wiki/Parallel_coordinates

⁵<https://en.wikipedia.org/wiki/Biplot>

✍ Copy the above figure, and answer the following questions in the report. You may search on the Internet to answer the questions.

Q1 What's the physical meaning the vector corresponded to each variable? Explain it in one sentence.

Q2 What are the factors closely related to happiness according to this biplot? Write down your answer and use one more sentence to explain why.

4.3 Visualizing Classification Result

When you struggle with the pure result of your classifier, a good way to debug is by visualization it.

4.3.1 Visualize SVM

⇒ Write script in `plot/SVM_visual.py` to visualize your submitted SVM classifier on **these two dimensions**: "Income" and "EducationLevel". You need to plot the classification boundary of your SVM classifier on these two dimensions. Note that all the categories variables are treated as discrete variables after preprocessing. *Hint*: You may learn from this [example](#).

✍ Copy down the above figure in your report.

5 Submission

Instructions for the submission are as follows. **Please follow them carefully.**

1. Test your code before submission. Any python scripts with syntax errors won't be accepted.
2. Zip all the required files into a single zipped file named `<student-id>_prj.zip`, where `<student-id>` should be replaced with your own student ID. e.g., `1155012345_prj.zip`. The submitted zip file should have the same structure as the provided `prj.zip`. Please do not change the names of provided files.
3. Submit the zipped file `<student-id>_prj.zip` to `cuhkcsci3320@gmail.com` before the deadline 23:59 on Tuesday, May. 02, 2017.

References

- [1] "Logistic regression scikit-learn documentation," http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.
- [2] "Naïve bayes scikit-learn documentation," http://scikit-learn.org/stable/modules/naive_bayes.html.
- [3] "SVM scikit-learn documentation," <http://scikit-learn.org/stable/modules/svm.html>.
- [4] "Random forest scikit-learn documentation," <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>.