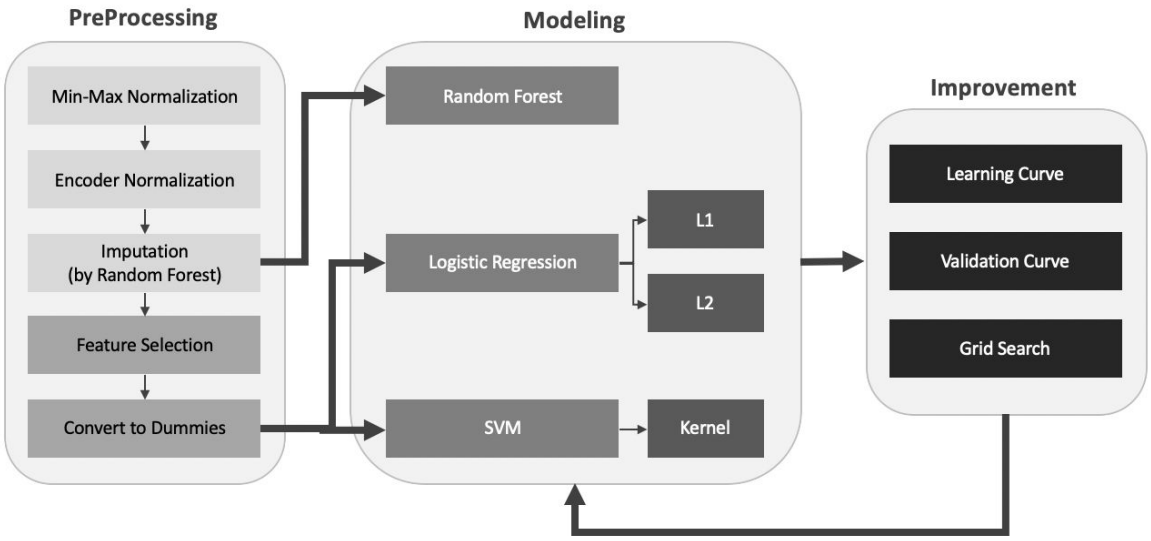
CISC 6930 Data Mining
Spring 2019
Team 5 Members: Chenhao Yang, Ronghan Liu, Qiangwen Xu, Zhiyu Xu, Yuqi Fu
Final Project Written Report
Project Link: [fordhamcisc69302019springfinalproject]

Overview

The final project uses a real-world dataset representing different people's income level. The dataset provides 14 attributes that correlate the income level status, such as marital status, family status, sex, etc. The purpose of our project is to establish an accurate model, based on the data being provided, to predict the income level. Our project uses various data mining algorithms, including SVM, Random Forest with interpretations into Python language. The whole project involves three stages: PreProcessing, Modeling, and Improvement.



Data Pre-processing

1. Normalization

Features from the initial dataset have various value ranges. For example, we can see the feature "education_num" distributing from 1 to 16. Meanwhile, "hours_per_week"

distributes from 1 to 99. Features with a wider value range would have greater influence in the following modeling.

To address this problem, we apply normalization with the min-max method and encoder method onto the initial dataset.

a. Min-max

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

For the features with continuous numeric values, we choose min-max normalization to compress its range into [0,1].

Features adopted:

- 'Capital_gain'
- 'Capital_loss'
- 'fnlwgt'

b. Encoder

For the features with string classes, we encode them with a value between 0 and [n_classes-1]. In result, their initial string classes are converted to integers, which is more suitable for further modeling.

Features adopted:

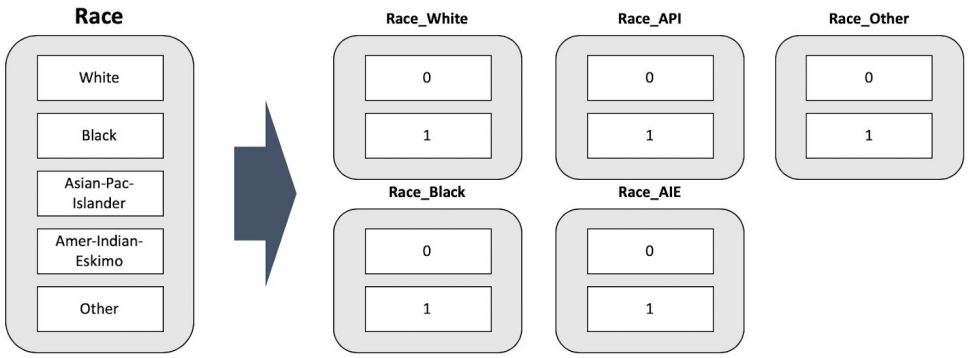
- 'Workclass',
- 'Education',
- 'Marital_status',
- 'Occupation',
- 'Relationship',
- 'Race',
- 'Sex',
- 'Native_country'

c. Dummy

The Encoder is not enough when implementing models. Some models like SVM requires further normalization to make sure it can treat every feature equally. We need to convert categorical variables into dummy/indicator variables, which contains only 0 or 1 class for each feature.

For example, when dealing with feature “race”, which contains 5 classes: ' White', ' Black', ' Asian-Pac-Islander', ' Amer-Indian-Eskimo', and ' Other'. It will be

extended into 5 new features (race_0 to race_4) each of which contains 2 classes 1 or 0.

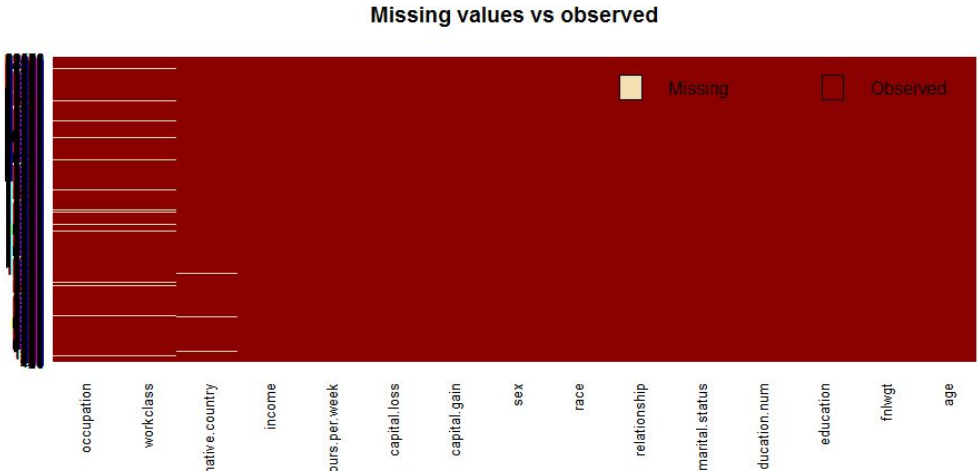


Based on our feature selection work (explained later), we need to covert only following features:

- 'Workclass'
- 'Marital_status'
- 'Race'
- 'Native_country'

2. Imputation

By scanning through the dataset, we find some missing-value issues needing to be addressed in the pre-processing phase.



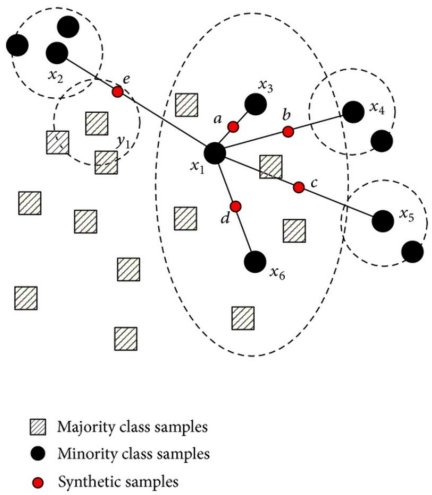
Approximate 7%(2399/32561) of the total data has missing value. The missing values are all in three features: *workclass*, *occupation*, and *native_countries*. Meanwhile, other features are unabridged.

Thus, we can implement classification models to predict those missing values based on the data we have. Considering the work time, we pick Random Forest as our prediction model to do imputation.

3. Dealing with the Imbalanced Data

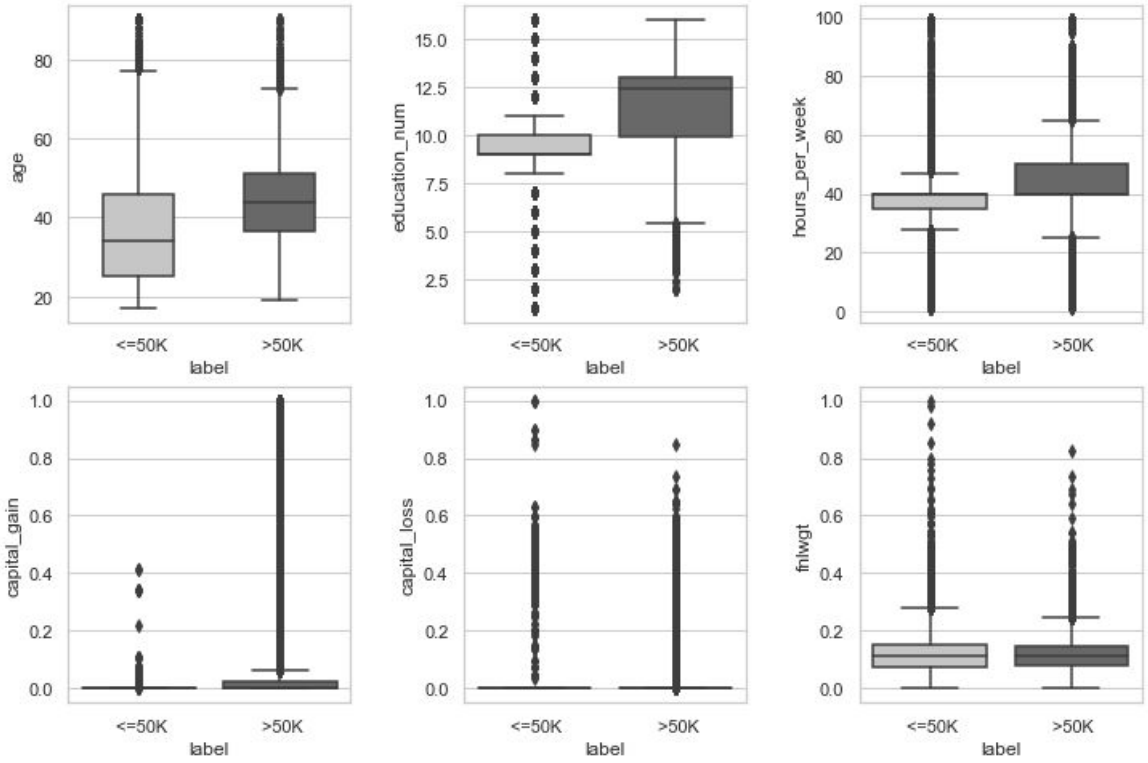
By counting the entries of two different labels (“>=50K”/ “<50K”), we find that our training dataset is imbalanced, which will generate a negative influence on the implementation of models, such as logistic regression and SVM. If the dataset is imbalanced, the model will be biased.

To deal with this problem, we take the oversampling method by import SMOTE module.



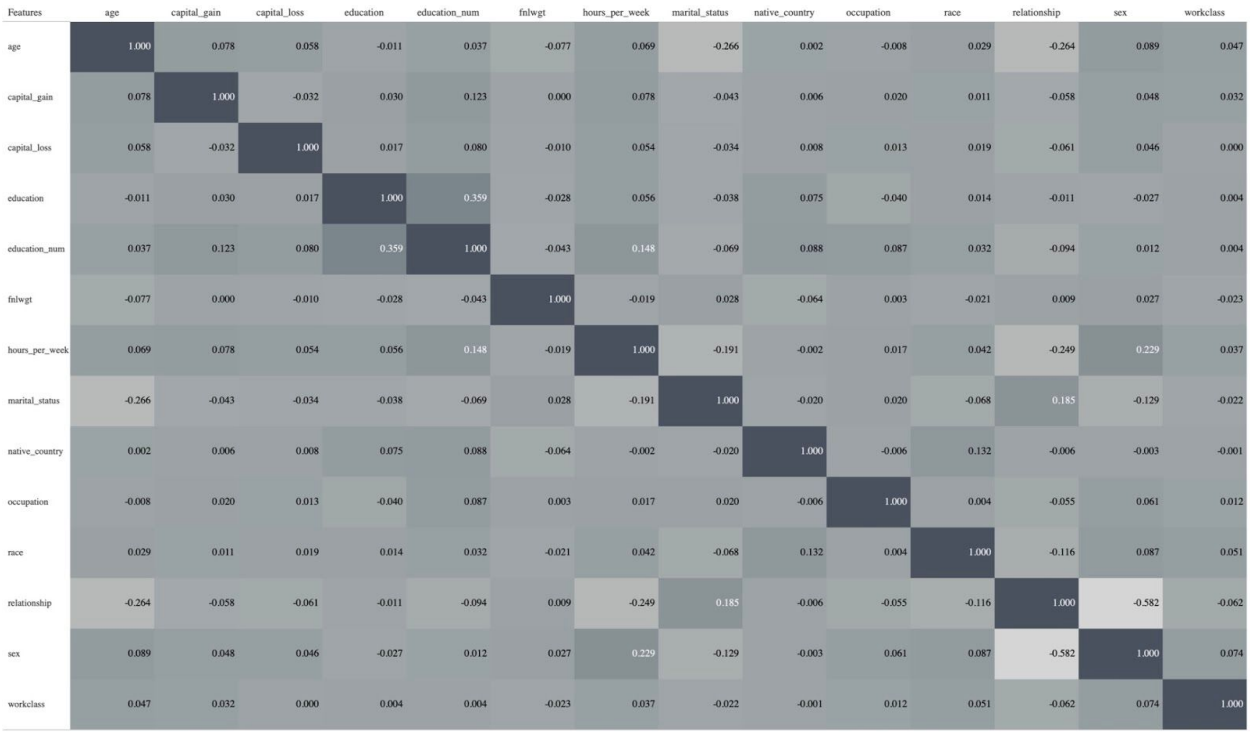
After the oversampling process, we get a balanced dataset by joining initial training dataset and new dataset generated from initial minority class samples.

Eventually, we get pre-processed train dataset through normalization (min-max, encoder, dummies), imputation, oversampling. Here are some descriptions of our train set:



Models

1. Feature selection



Features	age	capital_gain	capital_loss	education	education_num	fnlwgt	hours_per_week	marital_status	native_country	occupation	race	relationship	sex	workclass
age	1.000	0.078	0.058	-0.011	0.037	-0.077	0.069	-0.266	0.002	-0.008	0.029	-0.264	0.089	0.047
capital_gain	0.078	1.000	-0.032	0.030	0.123	0.000	0.078	-0.043	0.006	0.020	0.011	-0.058	0.048	0.032
capital_loss	0.058	-0.032	1.000	0.017	0.080	-0.010	0.054	-0.034	0.008	0.013	0.019	-0.061	0.046	0.000
education	-0.011	0.030	0.017	1.000	0.359	-0.028	0.056	-0.038	0.075	-0.040	0.014	-0.011	-0.027	0.004
education_num	0.037	0.123	0.080	0.359	1.000	-0.043	0.148	-0.069	0.088	0.087	0.032	-0.094	0.012	0.004
fnlwgt	-0.077	0.000	-0.010	-0.028	-0.043	1.000	-0.019	0.028	-0.064	0.003	-0.021	0.009	0.027	-0.023
hours_per_week	0.069	0.078	0.054	0.056	0.148	-0.019	1.000	-0.191	-0.002	0.017	0.042	-0.249	0.229	0.037
marital_status	-0.266	-0.043	-0.034	-0.038	-0.069	0.028	-0.191	1.000	-0.020	0.020	-0.068	0.185	-0.129	-0.022
native_country	0.002	0.006	0.008	0.075	0.088	-0.064	-0.002	-0.020	1.000	-0.006	0.132	-0.006	-0.003	-0.001
occupation	-0.008	0.020	0.013	-0.040	0.087	0.003	0.017	0.020	-0.006	1.000	0.004	-0.055	0.061	0.012
race	0.029	0.011	0.019	0.014	0.032	-0.021	0.042	-0.068	0.132	0.004	1.000	-0.116	0.087	0.051
relationship	-0.264	-0.058	-0.061	-0.011	-0.094	0.009	-0.249	0.185	-0.006	-0.055	-0.116	1.000	-0.582	-0.062
sex	0.089	0.048	0.046	-0.027	0.012	0.027	0.229	-0.129	-0.003	0.061	0.087	-0.582	1.000	0.074
workclass	0.047	0.032	0.000	0.004	0.004	-0.023	0.037	-0.022	-0.001	0.012	0.051	-0.062	0.074	1.000

In the picture above, the closer the feature's absolute value is to 1, the stronger the feature represents in the correlation analysis. With the results from the correlation analysis, we find out that the correlation between "education" and "education_num" is too high. Thus, we decide to drop the feature "education" because "education_num" is the better performing feature in the model.

To further determine which feature in the following four pairs performs better, we implement a/b-testing onto it:

Drop Feature	Accuracy score
without any deleting	**0.8407564724919094**
without_education	0.8451860841423948
without_education_num	0.8449838187702267

without_marital_status	0.84043284789644
without_relationship	0.8411812297734628
without_sex	0.8400485436893202
without_hours	0.836913430420712
without_workclass	0.8359163473235472
without_occupation	0.8387540453074434

2. Model Implementation

- **Logistic Regression**

  Parameter after tuning:

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    4.9s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   20.2s
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed:   21.5s finished
{'classifier__C': 1.623776739188721, 'classifier__penalty': 'l1', 'classifier__solver': 'liblinear'}
```

L1 logistic regression	c = 1	c = 2	c = 3	c = 4	c = 5	c = 6
train_scores	0.85033879	0.85035902	0.85044498	0.85034385	0.85043993	0.8504197
cv socore	0.84668285	0.84692557	0.84676375	0.84680421	0.84680421	0.84664239
C = 2						
cv accu	0.841019417					
test accu	0.808795529					
L2 logistic regression	c = 1	c = 2	c = 3	c = 4	c = 5	c = 6
train_scores	0.82365494	0.82369033	0.82378135	0.82369539	0.82372067	0.82372067
cv socore	0.82247168	0.82259304	0.82257282	0.82265372	0.82259304	0.82265372
C = 4						
cv accu	0.82032767					
test accu	0.791536146					

- **Random Forest**

  We don't need to operate feature selection on a random forest model. And we use the validation curve and grid search to adjust the parameter. Max-depth of

the tree is the most important parameter. We choose 38, get 0.85 in cross-validation accuracy and 0.84 in test accuracy.

The parameter after tuning:

```
Reloaded modules: __mp_main__
{'max_features': 'log2', 'n_estimators': 700}

In [3]: runfile('/Users/lordxuzhiyu/Desktop/Final
Project/Code')
{'max_features': 'log2', 'n_estimators': 700}
```

Random forest with logs2 max feature	max_depth = 1	max_depth = 2	max_depth = 3	max_depth = 4	max_depth = 5	max_depth = 6
train_scores	0.77587086	0.79500587	0.80015868	0.8187983	0.83130386	0.84097258
test socore	0.7762257	0.79341914	0.79801104	0.81746928	0.82734035	0.8375957
max_depth	27					
cross_validation	0.858937543469485					
test	0.8501934770591487					

- **SVM (RBF Kernel)**

  We have tested 4 kinds of kernel functions:

  a. Linear
  b. Poly
  c. RBF
  d. Sigmoid.

  Only the RBF kernel displays good performance. The parameter adjustment to SVM has no significant influence. We get:

SVM RBF	c = 1	c = 2	c = 3	c = 4	c = 5	c = 6
train_scores	0.82423645	0.83144721	0.83568467	0.8384557	0.84079187	0.84269316
cv socore	0.81761731	0.82216828	0.82572816	0.82750809	0.8285801	0.83005663
C = 9						
cv accu	0.829484117					
test accu	0.828755937					

- **KNN**

  We also tried KNN classifier even we don't think it's reasonable for this case.

The necessary assumption before implementing KNN is that we can treat the distance equally between different classes in categorical features.

For example, in “workclass”, we set the same distances between those two pairs: “Never-worked” and “Federal-gov”, “Self-emp-not-inc” and “Private”.

It’s fairly strong speculation which we can not easily conduct from the information we’ve gotten so far. Therefore, we provide KNN classifier here only for reference and comparison to other models. It’s not necessary to take its results seriously.

KNN	k = 1	k = 2	k = 3	k = 4	k = 5	k = 6
train_scores	0.99975728	0.92766485	0.90986044	0.88939624	0.87981392	0.87033778
cv socore	0.84021036	0.8111246	0.83106796	0.82038835	0.82554612	0.81933657
k = 1						
cv accu	0.835781					
test	0.763466618					

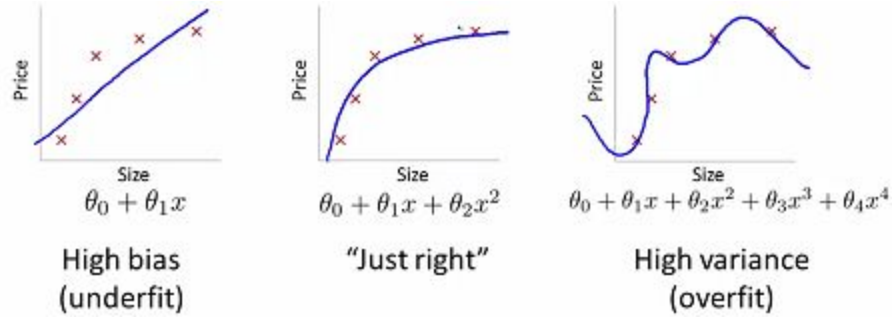
< Accuracy For Different K value in KNN >

Testing and Improvement

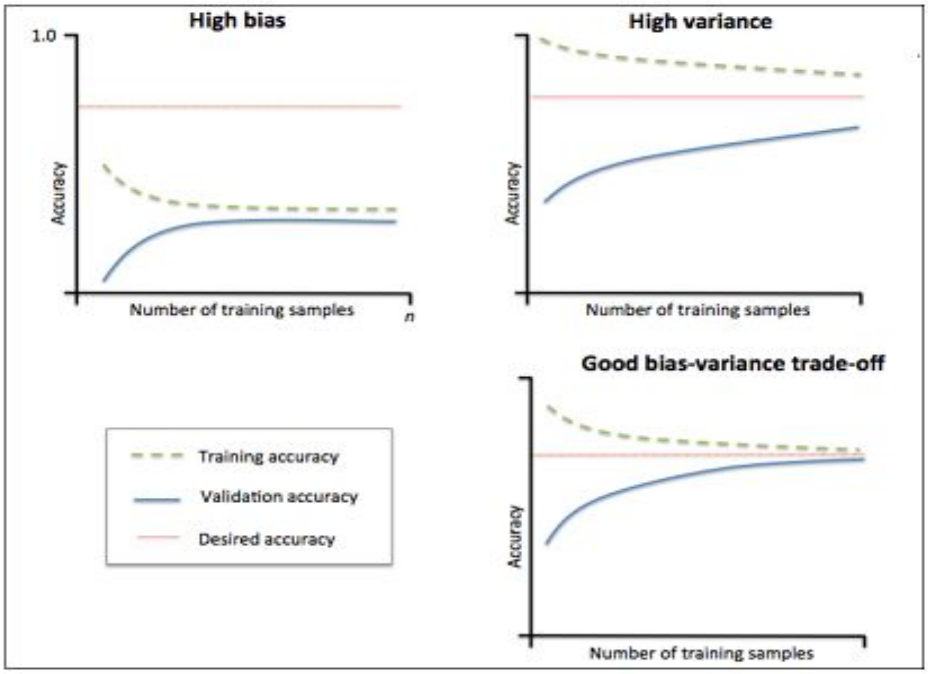
In this part, we are using different techniques to do the following things:

1. Evaluation

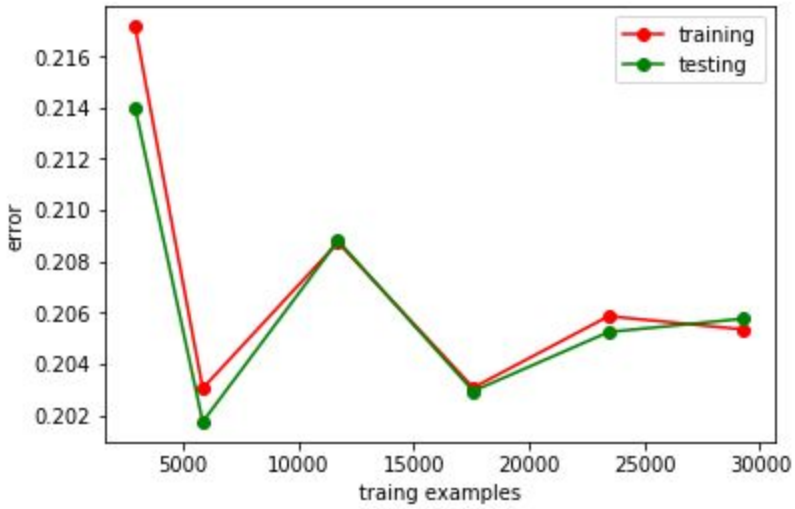
When running a learning algorithm, if the performance of the algorithm is not ideal, then there would be two situations: either the deviation is large or the variance is large. In other words, the situation is either an under-fitting or an overfitting problem. In the process of modeling, the model is often not fitted enough or over-fitting, but how to analyze the model for underfitting and overfitting? Underfitting and overfitting are typically analyzed by the learning curve and the evaluation curve of the model, and propose solutions in a practical application. The figure below is a theoretical way to identify underfitting and overfitting, as follows:



- **Learning Curve**
  If one model is too complicated, there would be many parameters that cause the model overfitting. To deal with the overfitting problem, we can reduce the complexity of the model, cross-check, by increasing the training data set. According to the above figure, we can gradually increase the model training set, observe the change on deviation and variance of the model, and determine whether this method is effective or not.

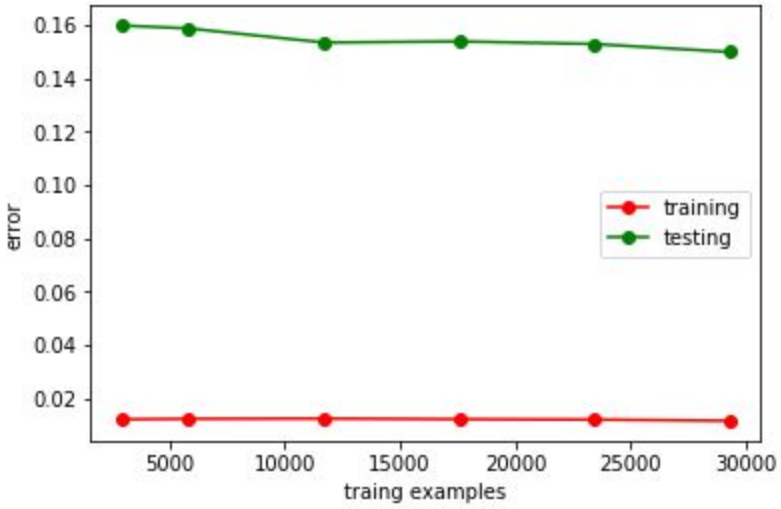


From the upper left of the figure above, the model's deviation is large, and the training and test-effect curves are very low-performing. Also, the accuracy is too low. It may be under-fitting, and the under-fitting problem is solved by increasing the model parameters, adding features or reducing the regularity Items, etc.



< Learning Curve for Logistic Regression >

As you can see from the graph, the learning curve for our logistic regression model. When the number of samples increases, the error rates for different dataset still works well.

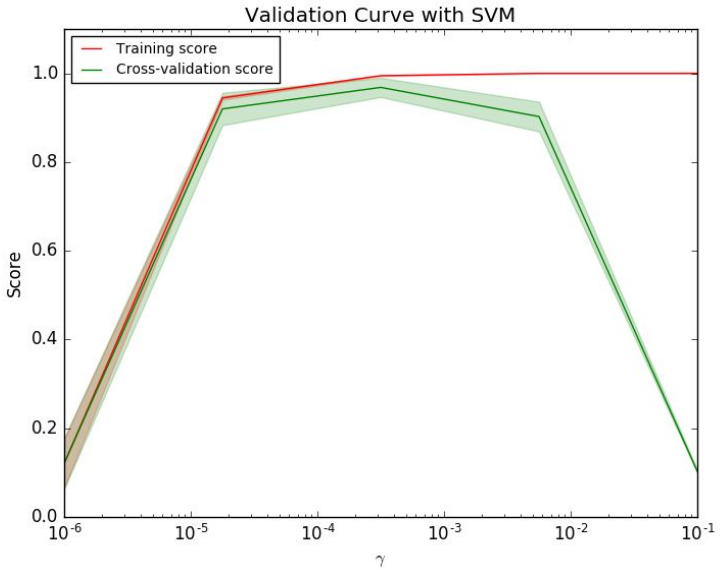


< Learning Curve for Random Forest >

This is the graph for our random forest model. The error rates of different datasets have a clear but still acceptable difference.

- **Validation Curve**
  The verification curve can adjust the parameters of the model to solve the over-fitting or under-fitting problem. The verification curve and the learning curve looks similar, but the meanings are different, which can be achieved by validation_curve. Models can also be optimized to increase the accuracy and generalization capabilities of the model. In the figure below (using a graph on the Internet), the model's effect changes as the parameters of the model change.

  If the training score and the verification score are both low, the model is in an under-fitting state. If the training score is high but the verification score is low, the model is in an overfitting state. If the training and verification scores are both high, then the model fits well. It is generally not possible to have both a low training score and a high verification score. We have listed the above three cases by changing the parameters of the SVM model: math.gamma:



< Validation Curve for SVM on 'Gamma' >

However, even though it's a very clear and thorough way to determine the best parameter for each model by using this curve, it may cost a lot of time and source by manually doing such. Therefore, we would implement another automatic method to figure out the best parameter setting for us.

- **Grid Search**

  Grid search is the process of performing hyperparameter tuning in order to determine the optimal values for a given model. This is significant since the performance of the entire model is based on the specified hyperparameter values.

  First, we need to import GridSearchCV from the <*sklearn*> library, a machine learning library for python. The estimator parameter of GridSearchCV requires the model we are using for the hyperparameter tuning process. For this example, we are using the RBF kernel of the Support Vector Regression model(SVR). The param_grid parameter requires a list of parameters and the range of values for each parameter of the specified estimator. The most significant parameters required when working with the RBF kernel of the SVR model are c, gamma, and epsilon. A list of values to choose from should be given to each hyperparameter of the model. You can change these values and experiment more to see which value ranges give better performance. A cross-validation process is performed in order to determine the hyperparameter value set which provides the best accuracy levels.

We then use the best set of hyperparameter values chosen in the grid search, in the actual model as shown above.

```
def LogisticRegressionTune():
    X, y = getData()
    pipe = Pipeline([('classifier' , RandomForestClassifier())])

    param_grid = [
        {'classifier' : [LogisticRegression()],
         'classifier__penalty' : ['l1', 'l2'],
        'classifier__C' : np.logspace(-4, 4, 20),
        'classifier__solver' : ['liblinear']},
        {'classifier' : [RandomForestClassifier()],
        'classifier__n_estimators' : list(range(10,101,10)),
        'classifier__max_features' : list(range(6,32,5))}
    ]
    clf = GridSearchCV(pipe, param_grid = param_grid, cv = 5, verbose=True, n_jobs=-1)
    clf.fit(X, y)
    # print(clf.best_params_)
    return clf.best_params_
```

< Logistic Regression Tune >

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    4.9s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   20.2s
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed:   21.5s finished
{'classifier__C': 1.623776739188721, 'classifier__penalty': 'l1', 'classifier__solver': 'liblinear'}
```

< Logistic Regression Parameter Result >

```
Reloaded modules: __mp_main__
{'max_features': 'log2', 'n_estimators': 700}

In [3]: runfile('/Users/lordxuzhiyu/Desktop/Final
Project/Code')
{'max_features': 'log2', 'n_estimators': 700}
```

< Random Forest Parameter Result >

SVM RBF Kernel Parameter Result:

Detailed classification report:

Best parameters set found on development set:

{'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}

Parameter set	Precision	Recall	F1-score	Support
0	1.00	1.00	1.00	89

1	0.97	1.00	0.98	90
2	0.99	0.98	0.98	92
3	1.00	0.99	0.99	93
4	1.00	1.00	1.00	76
5	0.99	0.98	0.99	108
6	0.99	1.00	0.99	89
7	0.99	1.00	0.99	78
8	1.00	0.98	0.99	92
9	0.99	0.99	0.99	92
Micro avg	0.99	0.99	0.99	899
Macro avg	0.99	0.99	0.99	899
Weighted avg	0.99	0.99	0.99	899

Best parameters set found on the development set:
{ ‘C’ : 10, ‘gamma’ : 0.001, ‘kernel’: ‘RBF’}

Grid scores on development set:

Grid scores	Parameter set
0.986 (+/-0.016)	C:1 gamma:0.001 kernel: RBF
0.959 (+/-0.029)	C:1 gamma:0.0001 kernel: RBF
0.988 (+/-0.017)	C:10 gamma:0.001 kernel: RBF
0.982 (+/-0.026)	C:10 gamma:0.0001 kernel: RBF
0.988 (+/-0.017)	C:100 gamma:0.001 kernel: RBF
0.982 (+/-0.025)	C:100 gamma:0.0001 kernel: RBF
0.988 (+/-0.017)	C:1000 gamma:0.001 kernel: RBF
0.982 (+/-0.025)	C:1000 gamma:0.0001 kernel: RBF
0.975 (+/-0.014)	C:1 gamma:0.001 kernel: RBF
0.975 (+/-0.014)	C:10 kernel: linear
0.975 (+/-0.014)	C:100 kernel: linear
0.975 (+/-0.014)	C:1000 kernel:linear