CMPUT 466 Mini-project final draft

Ziming Fu 1370023
Jiazhen Li 1501597

Introduction:

Surveys have shown that both in Canada and the U.S. It's the same reason why politicians love the word so much: When they saying "middle class," the vast majority of voters think it's about them, even though it might not be.

So what are some of the features shared by the population in the middle class? The median household income reached $57,617 in 2018, according to the latest census data. Our study is to determine whether a person's income exceeds 50k/year (the lower threshold for the entry of middle class) depends on the observation of features. This is a typical binary classification problem. Three different machine learning algorithms (Logistics regression, SVM, and Naive Bayes) with different hyper-parameters were studied with the project. We will analyze the general performance of these algorithms and evaluate the statistical significance of the following tests.

Dataset description:

Our dataset is a real-world "Census Income" dataset, collected and donated by UCI Data Mining and Visualization Silicon Graphics. More details on the resource link.
The dataset is a census data with 14 features and a target label: y ⇒ indicates whether a person's income exceeds 50k/year or not? (Binary label: "1" means yes; "0" means no).
There are two main types of features: categorical features and numerical features. The dataset contains a total of 48842 instances with some missing data scattered.
The specification of the features are listed:

age	continuous.
work-class	Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, etc.
fnlwgt	continuous.
education	Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, etc..
education-num	continuous.
marital-status	Married-civ-spouse, Divorced, Never-married, Separated, Widowed, etc.

occupation	Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, etc.
relationship	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
race	White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
sex	Female, Male.
capital-gain	continuous.
capital-loss	continuous.
hours-per-week	continuous.
native-country	United-States, Cambodia, England, Puerto-Rico, Canada, Germany, etc.

Fixing issues:

- Handling data rows with **missing values '?'**:
    1. Simply delete the samples from the dataset.
    2. Replace the missing value '?' with None.
    3. If the missing value '?' is category type, replace it with the most frequent instance from the category; If the missing value is numerical type, replace it with the mean/median of the distribution.

Approaches and Learning Methods:

Three approaches are used for this project: Logistics regression, Support vector machine(SVM), and Naive Bayes.

- **Logistic regression**
    - Description & why appropriate:
        - 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. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function.
        - $$\ell = \log_b \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$
    - Implementation:

        - Logistic regression is implemented via importing the scikit learn lib -- LogisticRegression. This implementation can fit the binary, One-vs-Rest, or multinomial logistic regression with optional, or Elastic-Net regularization.
    - Hyper-Parameters:
        - C (Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.): 0.0001, 0.01
        - Penalty type: 'none', 'l2'

- **Support vector machine(SVM)**
    - Description & why appropriate:
        - The support vector machine is a discriminative classifier formally defined by a separating hyperplane. A hyperplane is a line that divides the input variable space. In the SVM, the hyperplane is selected to optimally separate the points in the input variable space from their classes (0 level or 1 level). The SVM learning algorithm finds the coefficients that cause the hyperplane to best separate the classes using kernel trick.
        - SVM perfectly works with binary classification problems. And with many features this dataset, the benefit is that we can capture much more complex relationships between data points without having to perform difficult transformations on our own. SVM is effective in high dimensional spaces. Also, it uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
    - Implementation:
        - Support vector machine(SVM) is implemented via importing the scikit learn lib. The support vector machines in scikit-learn support both dense (numpy.ndarray and convertible to that by numpy.asarray) and sparse (any scipy.sparse) sample vectors as input. However, to use an SVM to make predictions for sparse data, it must have been fit on such data.
    - Hyper-Parameters:
        - Kernel types: ‘linear’, ‘poly’
        - C (Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared l2 penalty.): 0.0001, 1

- **Gaussian Naive Bayes:**
    - Description & why appropriate:
        - Gaussian Naive Bayes estimates a probabilistic model of the classification problem as described below. The prior for the coefficient is given by a spherical Gaussian. It is a supervised learning algorithm based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable.
        - Our target variable is discrete so we can use Naive Bayes methods. And since our dataset has continuous features, we shall use gaussian distribution for p(xi|y) instead of Bernoulli distribution. Also, It can be used to include regularization parameters in the estimation procedure.
        - $$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}}\, e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$
        - Also, Naive Bayes is the most straightforward and fast classification algorithm, which is suitable for a large chunk of data.
    - Implementation:
        - Gaussian Naive Bayes is implemented via importing the scikit learn lib - The Gaussian Naive Bayes building in the classifier.
    - Hyper-Parameters:
        - Priors: no priors, [0.1, 0.9]

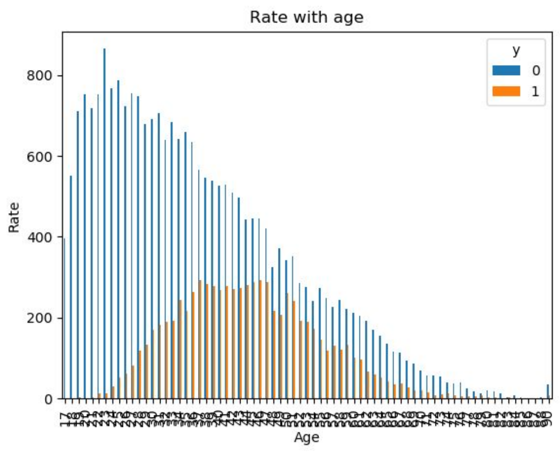
Experiment Design:

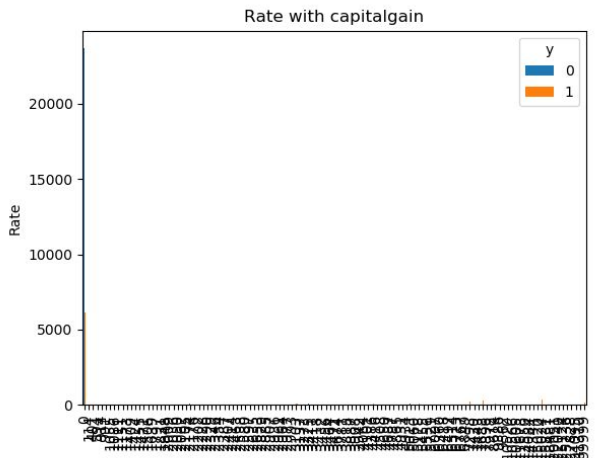
- **Data handling:**
    - **Fixing imbalanced data:**
        - Evaluate the numbers of data chunks separated by 2 different labels of the target column ie. 1 or 0. If there is a big difference in terms of size between the two numbers' percentage weight, the classes need to be balanced.

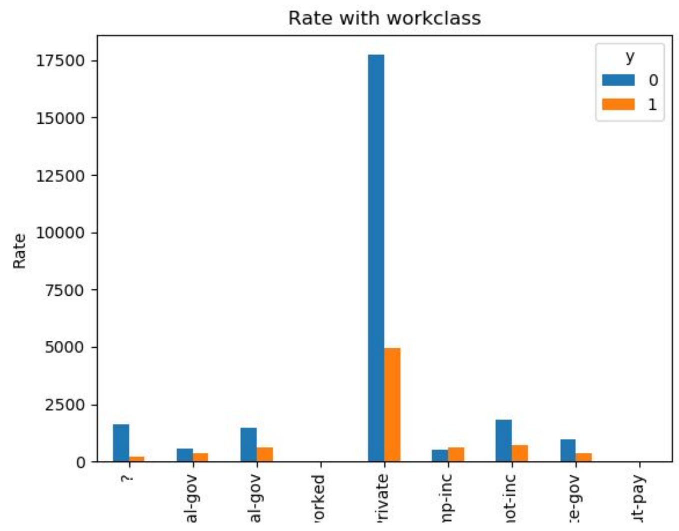
        - **Upsampling**: To balance the classes, we can simply create dummy copies from the minor class to up-sample them. Or we could use the SMOTE algorithm(Synthetic Minority Oversampling Technique) which chooses one of the k-nearest neighbors and uses it to create a similar but randomly tweaked new observations.

    ■ NOTE: Only up-sample data in the training set!

- **Feature selections**:
    - ❏ Use **Recursive feature elimination** to choose select best performing features. It is based on the idea to repeatedly construct a model and choose either the best or worst performing feature, setting the feature aside and then repeating the process with the rest of the features. This method is implemented by sklearn.feature_selection.
    - ❏ Check data set grouped by each feature to check sense and influence to the target variable. For example:







So ‘capital gain’ is not a good predictor for this classification problem, while ‘age’ and ‘class’ could be good predictors for this classification problem.

- **Training-test sets splitting:**
  - Typically, we split the dataset into 2 parts: training set, and test set. However, by partitioning the available data into 2sets, we drastically reduce the number of samples which can be used for learning the model, and the results can depend on a particular random choice for the pair of (train, validation) sets.
  - **K-fold cross-validation** was introduced to solve such a problem above and also prevent overfitting and generalize the model to seen data with low bias and variance. In *K*-fold cross-validation, the data is partitioned into *K* subsets. We then iteratively train on each set of (*K-1*) subsets and validate on the remaining subset, allowing each data point to be used as training data (*K-1*) times and validation data 1 time. (If the data is imbalanced, balance only the training set before each training)The model score is then computed as the average validation score across all *K* trials.
  - We used **K = 10** folds for cross-validation. It is a good choice for this problem.

- **Meta-parameter tuning and selection techniques:**
    - When evaluating different algorithms, or hyper-parameters, it is common to use **k-fold cross-validation** as mentioned in training -test datasets splitting. This allows you to train and test k models with different hyperparameters across the algorithms and collect their evaluation metrics, using their mean as an indication of the performance of the algorithm.

- **Statistical significance tests**:
    - **Null Hypothesis(H0)**:
      Our null hypothesis is that the performance metrics are equal, that any small gain, or loss, observed is not statistically significant.
    - Appropriate Statistical Test:
      Wilxon's test

  Since we need to compare the three algorithms, it is better to make significant tests pairwise to reject H0 more clearly.

  Wilxon's test is a good choice for this since it is a non-parametric test and the distribution of the differences between the two samples cannot be assumed to be normally distributed.
  The key reason for not using the paired t-test is an assumption of the paired Student's t-test has been violated. Namely, the observations in each sample are not independent. Also, each feature is not independent. As part of the k-fold cross-validation procedure, a given observation will be used in the training dataset (k-1) times. This means that the estimated skill scores are dependent, not independent, and in turn that the calculation of the t-statistic in the test will be misleadingly wrong along with any interpretations of the statistic and p-value.

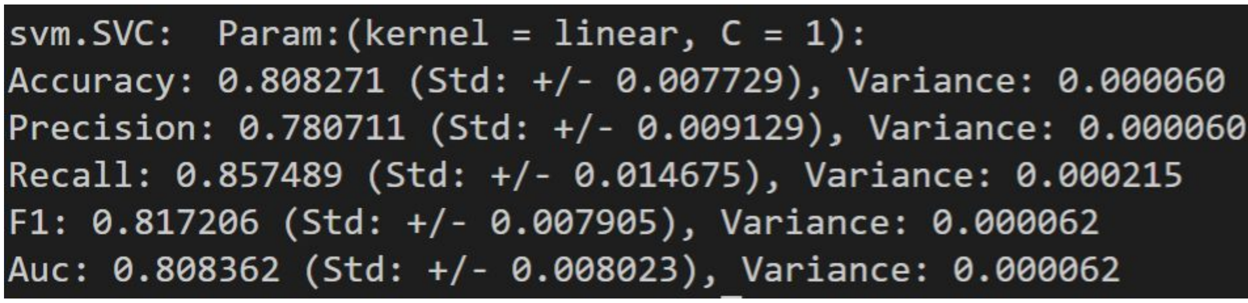
- **Models Results Comparison**:
    - Check the Significant test result above. Use confusion matrix to calculate **Accuracy**, **Precision**, **Recall**.
    - **F1 function**: F1=(2×Precision×Recall)/(Precision+Recall).
    - Draw **ROC**(Receiver Operating Characteristic) curve: the two main indicators in the ROC curve are Real rate and false-positive rate.
    - Calculate **AUC**(Area under the curve).

Algorithm performance & analysis:

1. **Support vector machine(SVM):**

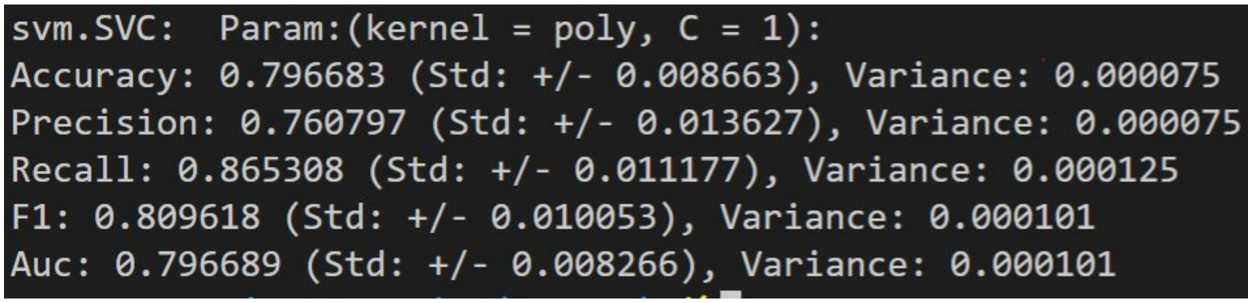
   Hyperparameters Tuning: Support vector machine(SVM) Param(kernel = linear, C = 1) returns the best Scores.

- ❏ svm.SVC Param(kernel = linear, C = 1)



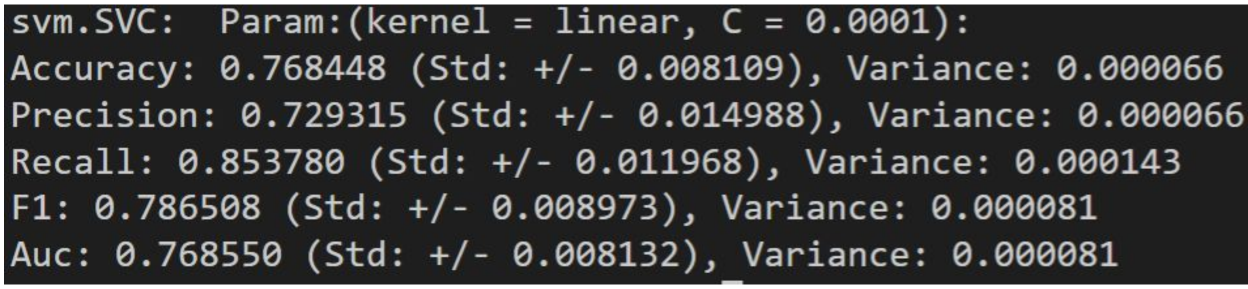
```
svm.SVC:  Param:(kernel = linear, C = 1):
Accuracy: 0.808271 (Std: +/- 0.007729), Variance: 0.000060
Precision: 0.780711 (Std: +/- 0.009129), Variance: 0.000060
Recall: 0.857489 (Std: +/- 0.014675), Variance: 0.000215
F1: 0.817206 (Std: +/- 0.007905), Variance: 0.000062
Auc: 0.808362 (Std: +/- 0.008023), Variance: 0.000062
```

- ❏ svm.SVC Param(kernel = poly, C = 1)



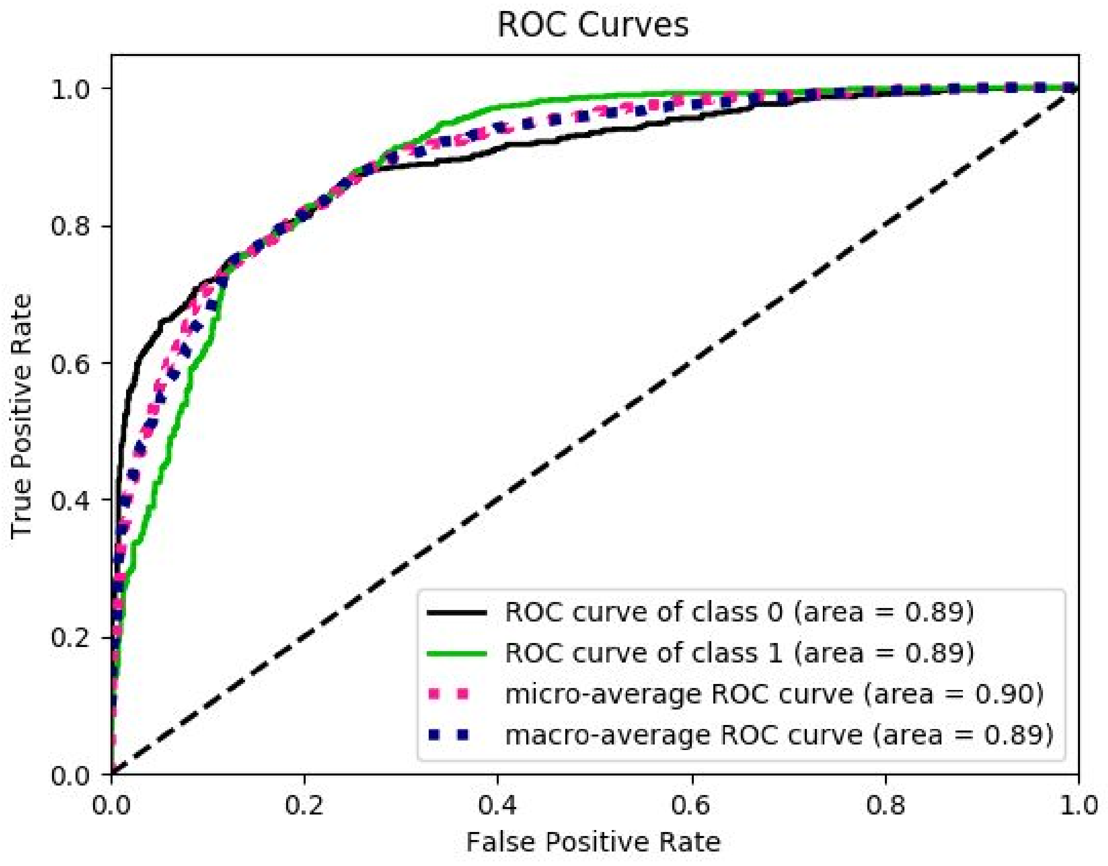
```
svm.SVC:  Param:(kernel = poly, C = 1):
Accuracy: 0.796683 (Std: +/- 0.008663), Variance: 0.000075
Precision: 0.760797 (Std: +/- 0.013627), Variance: 0.000075
Recall: 0.865308 (Std: +/- 0.011177), Variance: 0.000125
F1: 0.809618 (Std: +/- 0.010053), Variance: 0.000101
Auc: 0.796689 (Std: +/- 0.008266), Variance: 0.000101
```

- ❏ svm.SVC Param(kernel = linear, C = 0.0001)



```
svm.SVC:  Param:(kernel = linear, C = 0.0001):
Accuracy: 0.768448 (Std: +/- 0.008109), Variance: 0.000066
Precision: 0.729315 (Std: +/- 0.014988), Variance: 0.000066
Recall: 0.853780 (Std: +/- 0.011968), Variance: 0.000143
F1: 0.786508 (Std: +/- 0.008973), Variance: 0.000081
Auc: 0.768550 (Std: +/- 0.008132), Variance: 0.000081
```

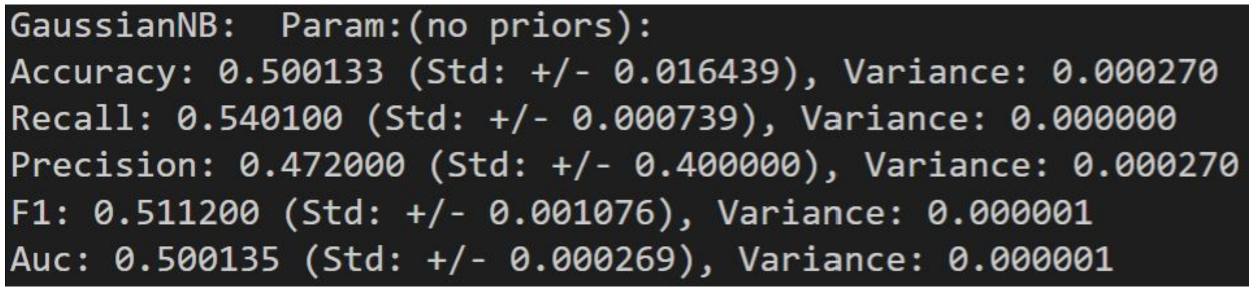
- ❏ Roc curve:



2. **Naive bayes:**

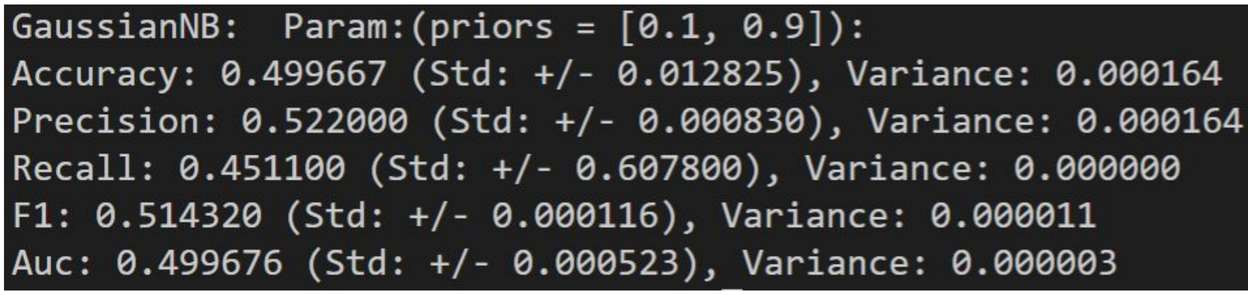
   Hyperparameters Tuning: GaussianNB Param(no priors) returns the best Scores.

   ❏ GaussianNB Param(no priors)



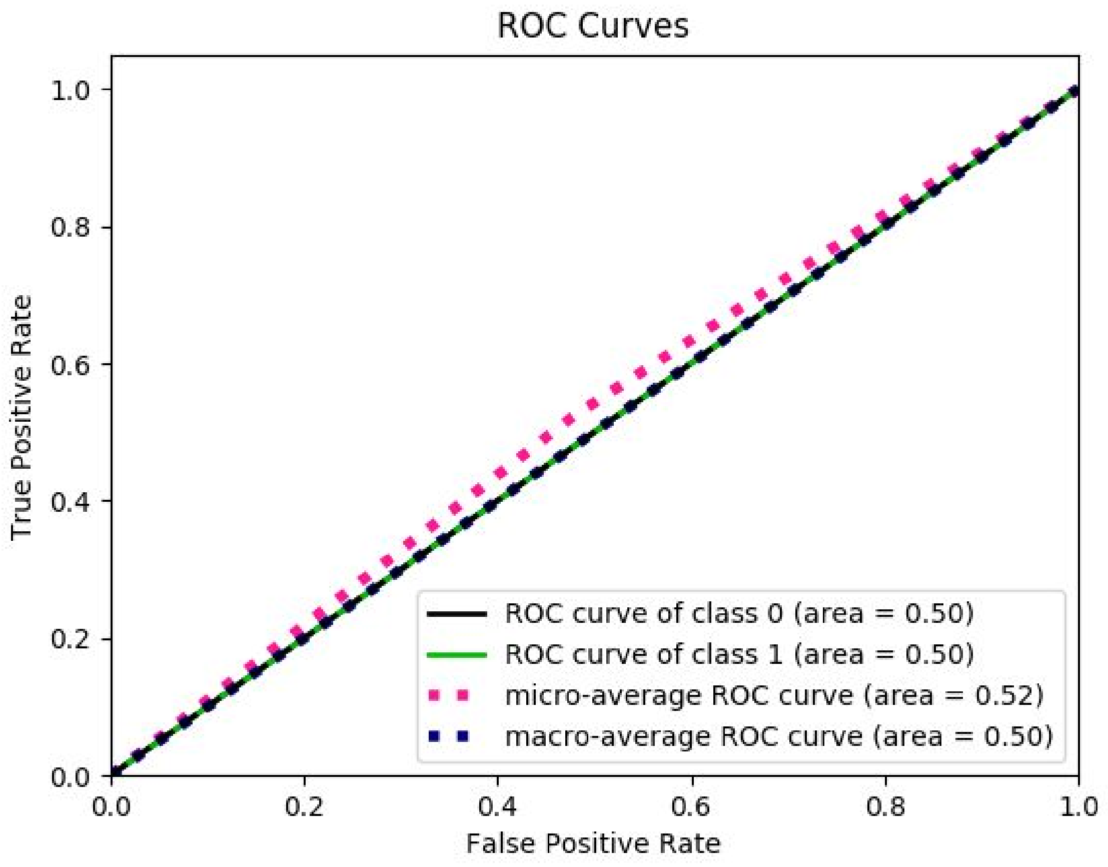
```
GaussianNB:  Param:(no priors):
Accuracy: 0.500133 (Std: +/- 0.016439), Variance: 0.000270
Recall: 0.540100 (Std: +/- 0.000739), Variance: 0.000000
Precision: 0.472000 (Std: +/- 0.400000), Variance: 0.000270
F1: 0.511200 (Std: +/- 0.001076), Variance: 0.000001
Auc: 0.500135 (Std: +/- 0.000269), Variance: 0.000001
```

   ❏ GaussianNB Param(priors = [0.1, 0.9])



```
GaussianNB:  Param:(priors = [0.1, 0.9]):
Accuracy: 0.499667 (Std: +/- 0.012825), Variance: 0.000164
Precision: 0.522000 (Std: +/- 0.000830), Variance: 0.000164
Recall: 0.451100 (Std: +/- 0.607800), Variance: 0.000000
F1: 0.514320 (Std: +/- 0.000116), Variance: 0.000011
Auc: 0.499676 (Std: +/- 0.000523), Variance: 0.000003
```

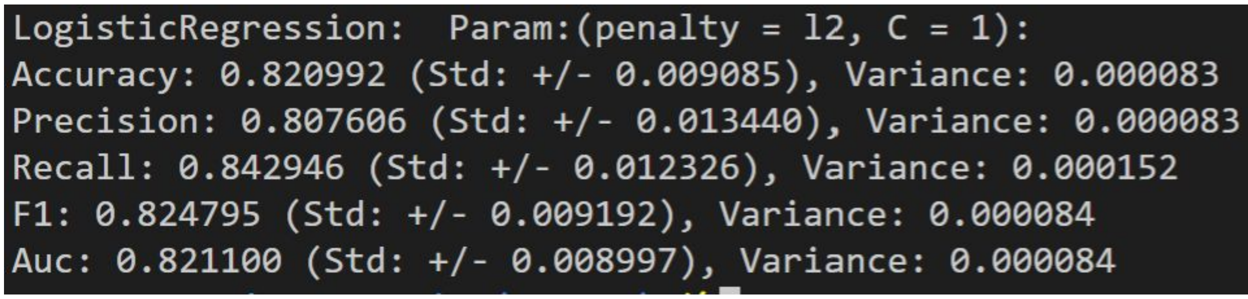
- ❏ Roc curve:



3. **Logistic Regression:**

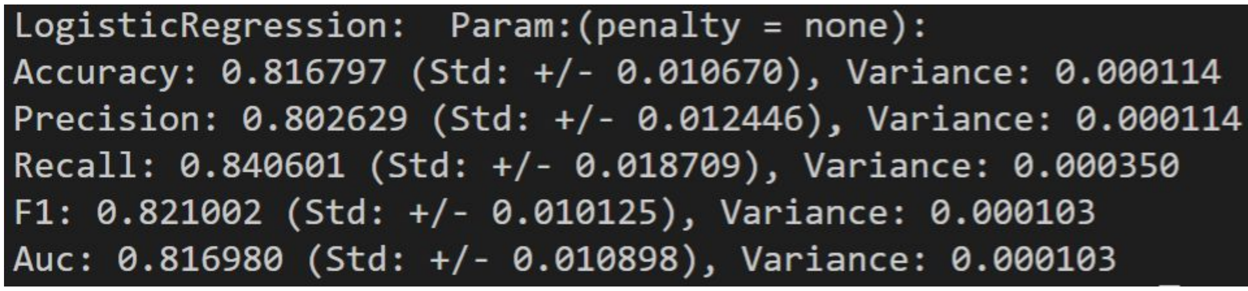
Hyperparameters Tuning: LogisticRegression Param(penalty = l2) returns the best Scores.

- ❏ LogisticRegression Param(penalty = l2, C = 1.0)



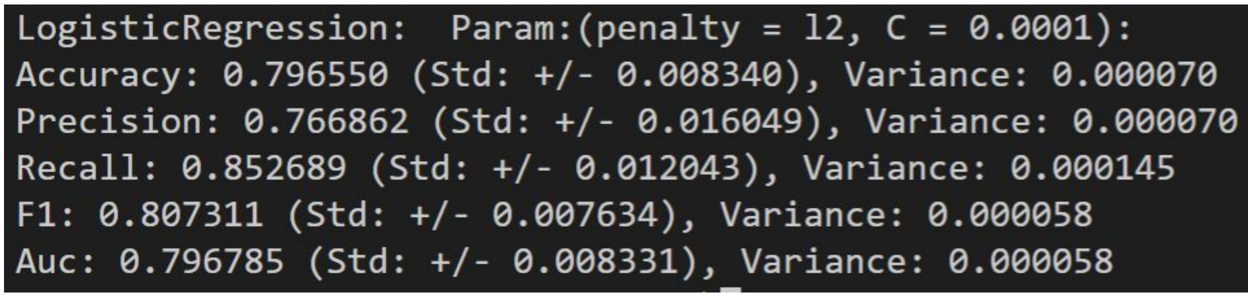
```
LogisticRegression:  Param:(penalty = l2, C = 1):
Accuracy: 0.820992 (Std: +/- 0.009085), Variance: 0.000083
Precision: 0.807606 (Std: +/- 0.013440), Variance: 0.000083
Recall: 0.842946 (Std: +/- 0.012326), Variance: 0.000152
F1: 0.824795 (Std: +/- 0.009192), Variance: 0.000084
Auc: 0.821100 (Std: +/- 0.008997), Variance: 0.000084
```

- ❑ LogisticRegression Param(penalty = none)



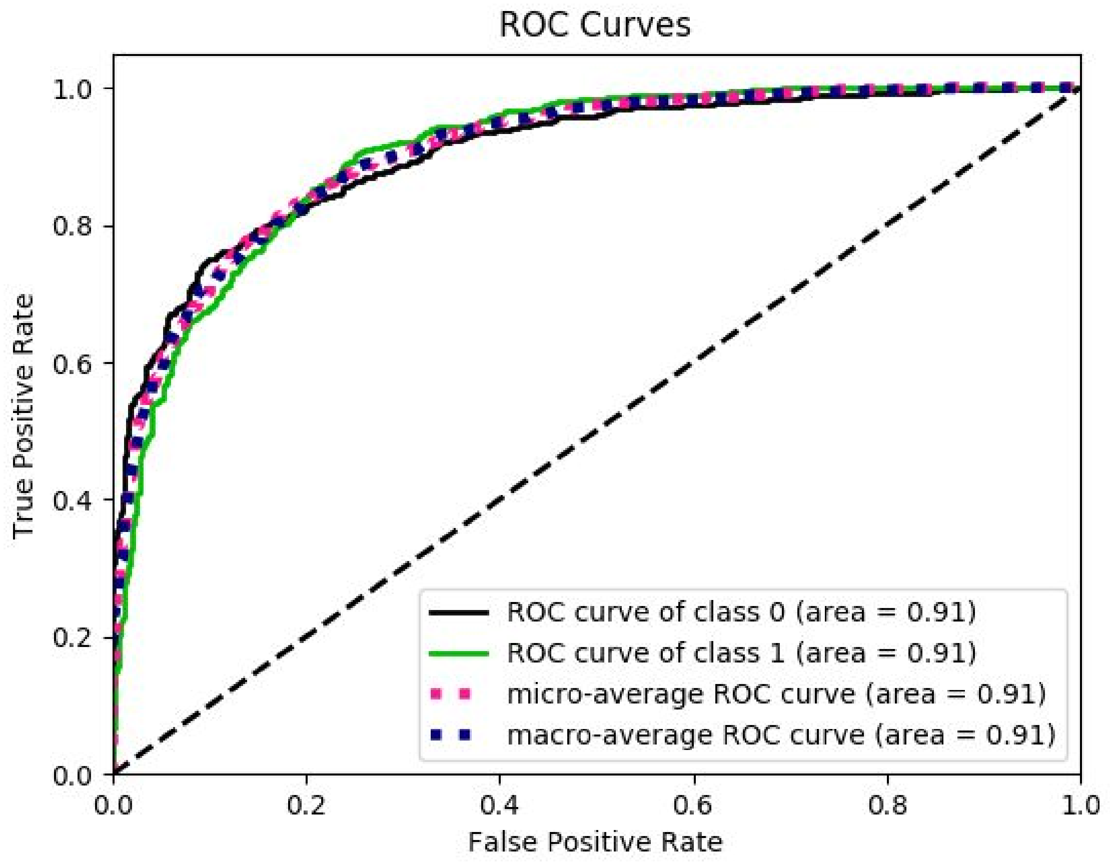
```
LogisticRegression:  Param:(penalty = none):
Accuracy: 0.816797 (Std: +/- 0.010670), Variance: 0.000114
Precision: 0.802629 (Std: +/- 0.012446), Variance: 0.000114
Recall: 0.840601 (Std: +/- 0.018709), Variance: 0.000350
F1: 0.821002 (Std: +/- 0.010125), Variance: 0.000103
Auc: 0.816980 (Std: +/- 0.010898), Variance: 0.000103
```

- ❑ LogisticRegression Param(penalty = l2, C = 0.0001)



```
LogisticRegression:  Param:(penalty = l2, C = 0.0001):
Accuracy: 0.796550 (Std: +/- 0.008340), Variance: 0.000070
Precision: 0.766862 (Std: +/- 0.016049), Variance: 0.000070
Recall: 0.852689 (Std: +/- 0.012043), Variance: 0.000145
F1: 0.807311 (Std: +/- 0.007634), Variance: 0.000058
Auc: 0.796785 (Std: +/- 0.008331), Variance: 0.000058
```

- ❑ Roc curve:



statistically significant test:

- **SVM & Naive Bayes:**
  W-value is 203410, p-value is 0.01

- **SVM & Logistic Regression:**
  W-value is 184735, p-value is 0.0

- **Naive Bayes & Logistic Regression:**
  W-value is 1923467, p-value is 0.01

- **Analysis:**
  As the 3 paired-tests all give p-values smaller than 0.05, we can reject H0 that there is enough evidence to prove 3 algorithms' performance is different.

Conclusion:

The Winning Algorithm for this project is the Logistic regression.
As we can see from the result of performance, Naive Bayes has a pretty low accuracy compared to the other two models, and the reason might be there are too many features in this dataset which drop the accuracy of prediction because naive Bayes makes a prediction by the probability.
By the ROC curve and other stats data, we can tell the logistic regression has a slight advantage over SVM while at the same time logistic regression takes a much shorter time for training model for this dataset than SVM. That is because the SVM requires the computation of a distance function between each point in the dataset and the storage of distance is a huge burden for memory which extended the training time. So for a large dataset like this, it is not a good idea to use SVM.

Reference:

- https://archive.ics.uci.edu/ml/datasets/
- https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
- https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- https://www.investopedia.com/terms/w/wilcoxon-test.asp
- https://elitedatascience.com/imbalanced-classes