Team name:
Team Speedwagon

Team members / email / contribution:
Andy Nguyen / andy_nguyen@csu.fullerton.edu / 33.3%
Rocio Salguero / salgueroroci@csu.fullerton.edu / 33.3%
Annie Chen / ann13ch3n@csu.fullerton.edu / 33.3%

Python3.6 used with Scikit-learn library, Pandas, Numpy, and Matplotlib

Project #2 Report

I. Best Classifier for Each Method

A. Naive Bayes

The best classifier for using Naive Bayes was using Gaussian Bayes because the gaussian distribution fit the values of the testing data better. Based on the scores on each of the models:

1. GaussianNB Score: 81.08
2. Multinomial Score: 79.29
3. Bernoulli Score: 73.28

Therefore for Naives Bayes method we used gaussian model.

B. Decision Tree

The best classifier for the decision tree method uses information gain to choose the best attributes for each of the nodes of the tree. Since the scikit-learn library offers two different methods to determine the best attributes for the tree, we attempted to use both the Gini index and the information gain method. The Gini index criterion yielded a slightly lower accuracy in comparison to the information gain criterion. In addition, other parameters were tried out over multiple trials to find the best classifier of the decision tree method. We tested out different parameters to try to increase the accuracy and to minimize the chances of the tree overfitting to the training dataset.

C. Multilayer perceptron (MLP)

The best classifier consists of using rectifier (ReLU) as an activation function, limiting the hidden layer size of 150, setting the regularization term (alpha) to 0.01, and setting iterations to 475.

II. Analyze classification results

A. Naive Bayes

Accuracy score using k-fold cross evaluation of k = 10 is 81.05%.

```
[0.80501302 0.81315104 0.80696615 0.80533854 0.80533854 0.81901042
 0.79654948 0.81835938 0.81830023 0.82025399]
GaussianNB Score: 81.08
Number of mislabeled points out of a total 30718 points : 5821, performance 81.05%
```

B. Decision Tree

Using the best classifier of the decision tree method, we found that the accuracy with k-fold cross validation of k=10 is 84.32%. The accuracy compares the actual Y values from the dataset with the Y values predicted by our classifier. This accuracy was the highest after trying out different parameters, such as the max depth and the minimum number of samples per leaf node.

C. Multilayer perceptron (MLP)

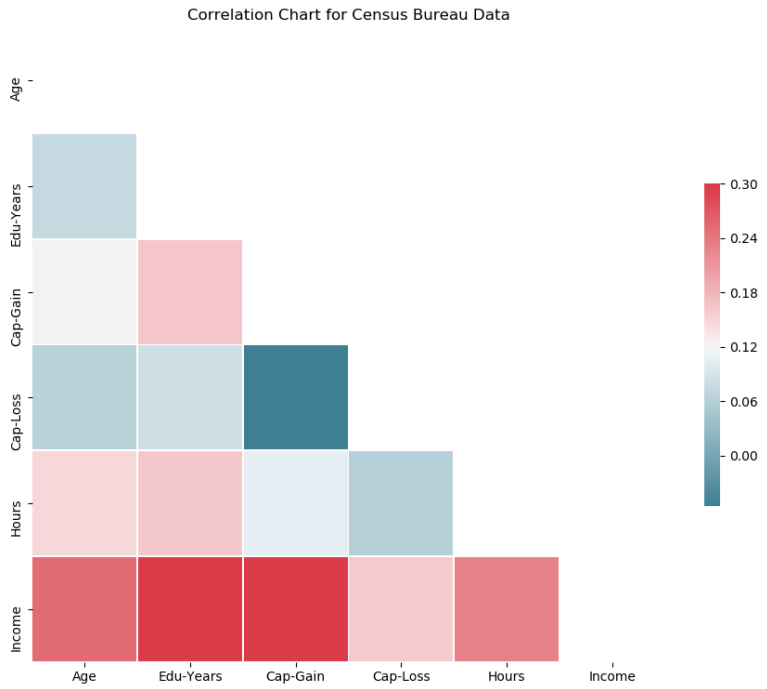
K-fold cross validation with k = 10 resulted in 84.43% accuracy when comparing actual Y's and predicted Y's. The precision and recall scored 84%, meaning the MLP model has a low false negative rate and low false positive rate. Since the f1 score, precision, and recall, are all high values, we can conclude the MLP is a great model to use for the dataset.

```
MLP Cross-Validation Score: 84.43
Accuracy (True Y vs Predicted Y:) 84.43
Classification Report (True Y vs Predicted Y):
             precision    recall  f1-score   support

          0       0.87      0.93      0.90     23068
          1       0.74      0.59      0.65      7650

avg / total       0.84      0.84      0.84     30718
```

III. Profile of a person most likely to make over $50k

Best profile:

Generally, people between the ages of 50 and 70, own a Master's degree, are married, are male, and work over 80 hours are most likely to make over $50k.



According to this chart, education years and capital gain are the two best predictors of income. The redder the squares are the closer the relation.

IV. Process of data analysis activities & results

We preprocessed the data before analyzing them with models. The columns were changed to numerical so the models can quickly read them. Several categories were combined such as “Ages > 21” and “Ages <30”. All columns that were changed to numerical values or were grouped can be found in the **Additional Code Reference** document. Since only 5% of the dataset was bad, it was safe to remove them without affecting results significantly. Initially, we were going to average the data, but it may add bias because the occupation value counts seemed unevenly distributed.

We also made bar charts to get visual representation of the distribution of each column in the dataset on how many make <=50K and > 50K. These charts are located under the Graphs folder included with this report. The charts were originally meant for us to visually see the distribution of the different categories of the dataset and to help us determine which columns in the dataset seemed to contribute to the income category. Ultimately, rather than disregarding and dropping certain categories of the dataset, we chose to use all the categories given.

A. Naive Bayes

Using naive bayes methods we used the Gaussian NB model. The model has prior of each class <=50K = 0 and >50K = 1.

```
 Model prior(Prior probabilities of the classes):  [0.75096035 0.24903965]
```

```
Model class: [0 1]
```

After fitting the model to the whole dataset we end up finding all the mean and std for all the attributes (columns) and the relation to classes.

Mean for <=50k starting with Age, Work, Edu-Lvl etc. :

```
Model mean:
 [[ 1.67092943  2.18922317 10.15692735  9.63650945  2.75073695  5.85083232
   1.64895093  0.61838911  0.05236692  0.06368129  1.17825559]
```

SD for <=50k:

```
Model SD:
 [[ 0.8362678   0.8451986  16.55088969  5.88733883  2.62141225 16.26979861
   2.38829569  0.23598403  0.08968013  0.16141202  0.42556847]
```

Mean for >50k starting with Age, Work, Edu-Lvl etc.:

```
[ 2.17856209  2.23137255 10.84627451 11.61934641  2.07777778  6.31764706
  0.71856209  0.85267974  0.68823529  0.27124183  1.53764706]]
```

SD for >50K:

```
[ 0.34354043  1.09627068  8.00800248  5.66007018  0.73761076 15.85857749
  2.44536788  0.12561702  1.90476357  0.73858475  0.42531474]]
```

In order to find the probability for each attribute for Income=>50K is using this formula and add the mean and SD:

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

With the scikit learn package we used the function **data.predict_proba(val)** to find the probability the value would be in <=50K or >50K. All the probabilities are in **Additional Code Reference**, and in order to find the best classifier we just choose the ones with the highest probability for >50K.

B. Decision Tree

We created the decision tree model using information gain as the criterion. The model was seeded with a random state of 100. Then, we played around with two optional parameters for the DecisionTreeClassifier from the scikit learn library: max_depth and min_sample_leaf. max_depth limited how deep the decision tree could go, and min_sample_leaf required that each leaf of the decision tree have at least a set number of samples to it. We set max_depth and min_sample_leaf to 10 and 10, respectively.

These two parameters were set such that we would minimize overfitting, which is a common problem with decision tree classifiers. These parameters are the basic ideas

behind pre-pruning a decision tree, or stopping the growth of the tree before it becomes overly complex and favoring a simpler tree. Unfortunately, the scikit learn library currently does not support any post-pruning processes, or they may have helped decrease the chances of overfitting more.

Using these parameters, we then used k-fold cross validation to create and test our model, which yielded an accuracy of 84.32%. We then used graphviz to create the visual representation of our decision tree. The final decision tree can be found under the Graphs folder, named "tree.png". From each node of the tree, "value" consists of two values. The first is the number of samples that fall under people who make <= 50K, and the second value is the number of samples that fall under people who make > 50K.

C. Multilayer perceptron (MLP)

We found the best classifier for the MLP by testing different parameters in a grid search. Parameters such as activation functions and a number of iterations were exhaustively tested until the best accuracy was found. Since the dataset is extensive and we needed to preprocess the data, the features were standardized to ensure the data size does not contribute to overfitting. Standardizing features consisted of scaling to variance and removing the mean with the StandardScaler scikit-learn library. ReLU was chosen over sigmoid because the accuracy improved by roughly 2%. We kept the hidden layer size at a relatively low amount to prevent over-specialization. The same concept applies to the regularization term and iterations since increasing them led to overfitting with an accuracy of 100%.