# CS311 Classification Report

by Yicheng 12210414

In this project, we are asked to learn a model on an Adult Census Income dataset, and the goal is to predict whether income exceeds \$50K/yr based on census data. The original data is from kaggle Adult Census Income. I try to utilize this chance to enhance my understanding on different models and how they perform, and try to figure out the reasons to their performance behaviors.

## Contents

# §1. Introduction to the dataset

The training data and the testing data has following attributes and its according types are as below.

| Attribute | Description | Type |
|---|---|---|
| **age** | the working age of each data sample | Numeric |
| **workclass** | type of work, where there are private, local government, etc. | Character |
| **fnlwgt** | the number of observational representatives of a sample in a state | Numeric |
| **education** | the level of education of each sample | Character |
| **education_num** | the schooling year of each sample | Numeric |
| **marital.status** | marital status of each sample | Character |
| **occupation** | the occupation of each sample | Character |
| **relationship** | the family relationship of each sample | Character |
| **race** | the race of each sample | Character |
| **gender** | the sex of each sample | Character |
| **capital.gain** | a capital gain is a profit that results from a disposition of a capital asset, such as stock, bond or real estate, where the amount realised on the disposition exceeds the purchase price | Numeric |
| **capital.loss** | capital loss is the difference between a lower selling price and a higher purchase price, resulting in a financial loss for each sample | Numeric |
| **hours_per_week** | sample weekly working hours | Numeric |
| **native_country** | the country where the sample is from | Character |

The training label is used to determine whether each person has income lower than and equal to \$50K/yr or larger than \$50K/yr.

| Attribute | Description | Type |
|---|---|---|
| **income** | whether income is greater than 50K and less than or equal to 50K | Boolean |

From the above data description, we can have a bulk understanding of the data we are processing. We are coping with a binary classification problems, with both numerical values and categorical values. From this aspect, we can predict that the classification boundaries ought not to be linear. Linearly-related methods may not perform greatly on this tasks. However, non-linear and self-learning methods may achieve higher hit-rate in this task.

### §1.1. Data Preprocessing

Some of the models do not accept categorical values and in some cases, standardization to data will lead to better performance. Our preprocessing is based on the models we used, the basic operations are:

- Load the data
- Categorize every attributes, and map each value to a numeric value
- Detect null or missing values (Some places use ? to represent unknown data)
- Standardization the data
- Use pipeline structure to find better parameters combination.

| Methods | Mapping | Standardization | Pipeline |
|---|---|---|---|
| **Decision Tree** | True | StandardScaler | False |
| **Naive Bayes** | True | False | False |
| **Nearest Neighbors Classifiers** | True | StandardScaler | False |
| **Support Vector Machines** | True | StandardScaler | True |
| **Ensemble Models** | True | StandardScaler | False |
| **Neural Networks** | True | StandardScaler<br>MinMaxScaler<br>RobustScaler | False |

## §2. Testing Environment

We are testing on two environments. All the experiments of Decision Tree, Naive Bayes, Nearest Neighbors Classifiers are tested on Platform (MacOS System):

```
System Software Overview:
  System Version: macOS 11.7.10 (20G1427)
  Kernel Version: Darwin 20.6.0
  Boot Volume: Macintosh HD - Data
```

```
  Boot Mode: Normal
  Secure Virtual Memory: Enabled
  System Integrity Protection: Enabled

Hardware Overview:
  Model Name: MacBook Pro
  Model Identifier: MacBookPro11,1
  Processor Name: Dual-Core Intel Core i5
  Processor Speed: 2.4 GHz
  Number of Processors: 1
  Total Number of Cores: 2
  L2 Cache (per Core): 256 KB
  L3 Cache: 3 MB
  Hyper-Threading Technology: Enabled
  Memory: 8 GB
  System Firmware Version: 478.0.0.0.0

Package Overview (Important Package):
  #Name                     #Version
  blas                      1.0
  glib                      2.78.4
  graphviz                  2.50.0
  intel-openmp              2021.4.0
  ipykernel                 6.29.3
  ipython                   8.12.2
  jupyter_core              5.7.2
  matplotlib                3.7.2
  mkl                       2021.4.0
  numpy                     1.24.3
  pandas                    2.0.3
  python                    3.8.19
  scikit-learn              1.3.0
  scipy                     1.10.1
  seaborn                   0.12.2
  zlib                      1.2.13
```

# §3. Models Introduction

## §3.1. Decision Tree

Decision Tree is a traditional model which can greatly suit our scenarios. It is opted for binary labels and categorical data.

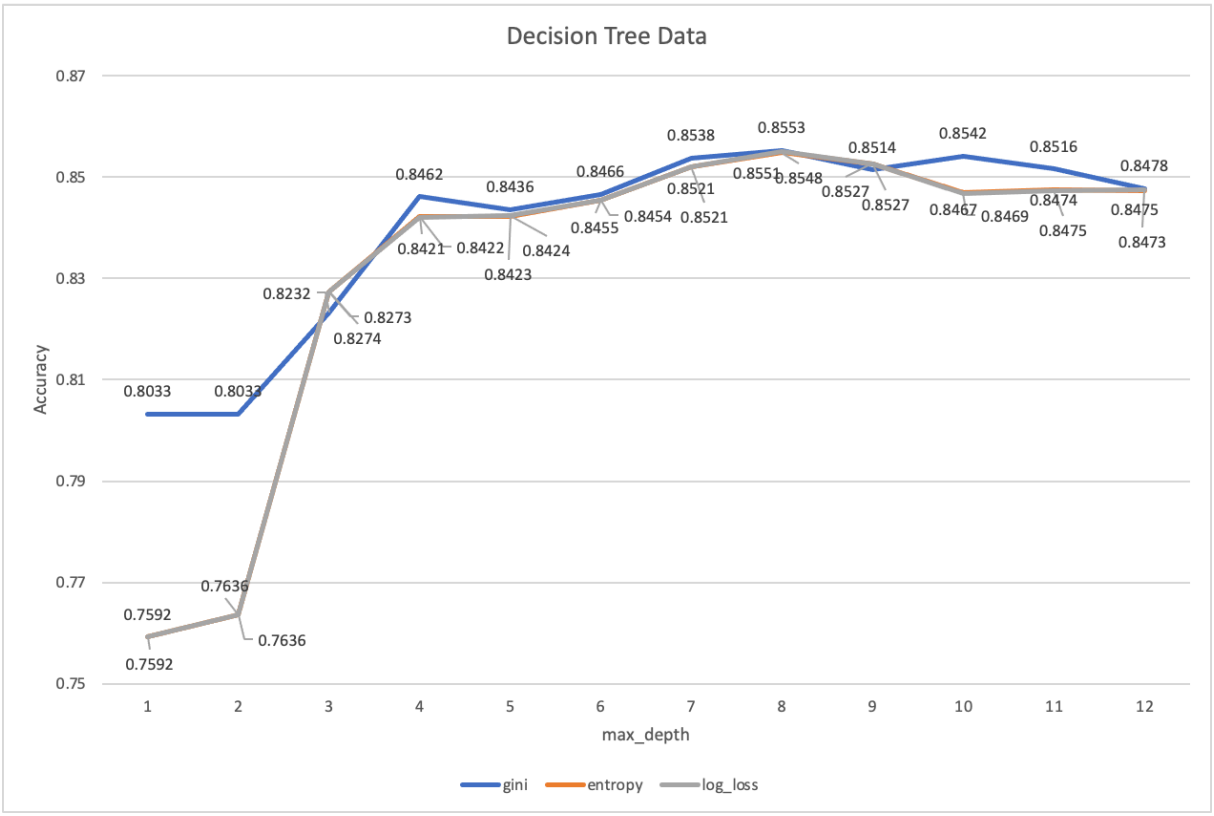


Figure 1: Decision Tree example

Given training vectors $x_i \in R^n$, $i = 1, ..., l$ and a label vector $y \in R^l$, a decision tree recursively partitions the feature space such that the samples with the same labels or similar target values are grouped together.

Let the data at node $m$ be represented by $Q_m$ with $n_m$ samples. For each candidate split $\theta = (j, t_m)$ consisting of a feature $j$ and threshold $t_m$, partition the data into $Q_m^{\text{left}}(\theta)$ and $Q_m^{\text{right}}(\theta)$ subsets.

$$Q_m^{\text{left}}(\theta) = \{(x, y) \mid x_j \leq t_m\}$$
$$Q_m^{\text{right}}(\theta) = Q_m \setminus Q_m^{\text{left}}(\theta)$$

The quality of a candidate split of node $m$ is then computed using an impurity function or loss function $H()$, the choice of which depends on the task being solved (classification or regression)

$$G(Q_m, \theta) = \frac{n_m^{\text{left}}}{n_m} H\big(Q_m^{\text{left}}(\theta)\big) + \frac{n_m^{\text{right}}}{n_m} H\big(Q_m^{\text{right}}(\theta)\big)$$

We try to select the parameters that minimizes the impurity $G(Q_m, \theta)$.[1]

The sklearn decision tree package only takes in numerical values, therefore we need to preprocess the data by mapping. The splitting criterion has three choices: `gini`, `entropy` and `log_loss`. Basically, the `gini` criterion is used to measure the impurity of the node.

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk})$$

The `log_loss` criterion is used to measure the logistic loss of the node and the `entropy` criterion is used to measure the information gain of the node..

$$H(Q_m) = -\sum_k p_{mk} \log p_{mk}$$

The realization is as follows:

```
# Decision Tree
from sklearn import tree
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
clf = tree.DecisionTreeClassifier(
    criterion='gini',
    splitter='best',
    max_depth=10
)
clf = clf.fit(X_train_nn, Y_train)
Y_test = clf.predict(X_test_nn)
accuracy = accuracy_score(test_label, Y_test)
print(f'Accuracy: {accuracy:.4f}')
print(classification_report(test_label, Y_test))
print(sns.heatmap(confusion_matrix(test_label, Y_test), annot=True, fmt='d'))
```

## §3.2. Naive Bayes

In `sklearn` toolkit, there are four realizations of Naive Bayes Classifiers tested in our experiments, each implemented in various scenarios. They are Gaussian Naive Bayes, Multinomial Naive Bayes, Complement Naive Bayes and Bernoulli Naive Bayes.

| **Feature** | **GaussianNB** | **MultinomialNB** | **ComplementNB** | **BernoulliNB** |
|---|---|---|---|---|
| **Use Case** | Features follow a Gaussian distribution | Multinomially distributed data, | Similar to MultinomialNB but for imbalanced data | Binary/ Boolean features |
| **Parameter Estimation** | Mean and variance for each feature | Relative frequency counting | The complement of each class to compute the model's weights. | Feature probabilities for binary features |
| **Common Applications** | Sensor data, real-valued data | Text classification (where the data are typically represented as word vector) | Text classification with class imbalance | Text classification with binary features |

| Feature | GaussianNB | MultinomialNB | ComplementNB | BernoulliNB |
|---|---|---|---|---|
| **Handling of Zero Counts** | Not applicable | Smoothing (Laplace or other) | Smoothing (Laplace or other) | Smoothing (Laplace or other) |
| **Pros** | Handles continuous data, simple and fast | Effective for text classification | Better performance with imbalanced data | Simple and effective for binary data, better on some datasets, especially those with shorter documents. |
| **Cons** | Assumes normal distribution of features | Not suitable for continuous data | More complex than MultinomialNB | Not suitable for non-binary data |

Theoretically, Complement Naive Bayes Classifier might has a great performance over Adult Census Income due to the categorical(discrete) features. The realization is as follows:

```
from sklearn.naive_bayes import GaussianNB, MultinomialNB, ComplementNB, BernoulliNB
gnb = GaussianNB()
y_test_gnb = gnb.fit(X_train, Y_train).predict(X_test)
accuracy_gnb = accuracy_score(test_label, y_test_gnb)
print(f'Gaussian NB Accuracy: {accuracy_gnb:.4f}')

mtb = MultinomialNB()
y_test_mtb = mtb.fit(X_train, Y_train).predict(X_test)
accuracy_mtb = accuracy_score(test_label, y_test_mtb)
print(f'Multinomial NB Accuracy: {accuracy_mtb:.4f}')

cmb = ComplementNB()
y_test_cmb = cmb.fit(X_train, Y_train).predict(X_test)
accuracy_cmb = accuracy_score(test_label, y_test_cmb)
print(f'Complement NB Accuracy: {accuracy_cmb:.4f}')

bnb = BernoulliNB()
y_test_bnb = bnb.fit(X_train_nn, Y_train).predict(X_test_nn)
accuracy_bnb = accuracy_score(test_label, y_test_bnb)
print(f'Bernoulli NB Accuracy: {accuracy_bnb:.4f}')
```

## §3.3. Nearest Neighbors Classifiers

There are in total two kinds of Nearest Neighbors Classifiers, ***k*-Neighbors Classifiers** and **Radius Neighbors Classifier**. The key of using nearest neighbor classifiers is to find the best k value (How many neighborhoods are there for all the data to be grouped into) or the best radius(What should be the radius of each neighborhood). This will delineate the clustering boundaries. However, in the case of Radius Neighbors Classifier there are outliers if the radius are set too small. In this case, we can only label the outliers, and the radius size is dependent

on whether the data is standardized. The $k$-Neighbors Classifier is more stable than the Radius Neighbors Classifier, and the accuracy is higher after standardization. The realization is as follows:

```
knn = KNeighborsClassifier(
    n_neighbors=80,
    weights='uniform',
    algorithm='auto',
    leaf_size=30,
    p=2,
    metric='minkowski',
    metric_params=None,
    n_jobs=None
)
knn.fit(X_train_nn, Y_train)
y_pred_knn = knn.predict(X_test_nn)

accuracy = accuracy_score(test_label, y_pred_knn)
print(f'KNN Accuracy: {accuracy:.4f}')

rnc = RadiusNeighborsClassifier(
    radius=2.25,
    weights='uniform',
    algorithm='auto',
    leaf_size=30,
    p=2,
    metric='minkowski',
    outlier_label=-1,
    metric_params=None,
    n_jobs=None
)
rnc.fit(X_train_nn, Y_train)
y_pred_rnc = rnc.predict(X_test_nn)

accuracy = accuracy_score(test_label, y_pred_rnc)
print(f'Radius Neighbors Accuracy: {accuracy:.4f}')
```

## §3.4. Support Vector Machines

Using linear support vector machine in such scenarios theoretically cannot produce good prediction results based on the intrinsic traits of the data. However, we can use different kernel like `rbf` to cope with such non-linear data. The realization is as follows:

```
from sklearn.svm import SVC, NuSVC, LinearSVC
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

param_grid = {
    'svc__C': [0.1, 1, 10, 100],
    'svc__gamma': [1, 0.1, 0.01, 0.001],
}
pipeline = make_pipeline(StandardScaler(), SVC(kernel='rbf'))
grid_search = GridSearchCV(pipeline, param_grid, refit=True, verbose=2, cv=5)
grid_search.fit(X_train, Y_train)
```

```
print("Best parameters found: ", grid_search.best_params_)
print("Best estimator: ", grid_search.best_estimator_) # Get result of svc_C = 1,
svc_gamma = 0.1

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
svc = SVC(
    C=1,
    kernel='rbf',
    degree=3,
    gamma=0.1,
    coef0=0.0,
    shrinking=True,
    probability=False,
    tol=0.001,
    cache_size=200,
    class_weight=None
)
svc.fit(X_train_nn, Y_train)
y_pred_svc = svc.predict(X_test_nn)
accuracy_svc = accuracy_score(test_label, y_pred_svc)
print(f'SVC Accuracy: {accuracy_svc:.4f}')
print(classification_report(test_label, y_pred_svc))
print(sns.heatmap(confusion_matrix(test_label, y_pred_svc), annot=True, fmt='d'))
```

## §3.5. Ensemble Models

Two very famous examples of ensemble methods are gradient-boosted trees and random forests. Ensemble models utilizes decision tree as weak learners. They are additive models whose prediction $\hat{y}_i = F_{M(X_i)} = \sum_{m=1}^{M} h_m(x_i)$. However, for classification problem, they still require a further mapping. This process is loss-dependent. In `sklearn` package, Histogram is used to preserve historical information, which will improve the basic gradient-boosted trees performance. The realization is as follows:

```
from sklearn.ensemble import HistGradientBoostingClassifier, RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
gbdt = HistGradientBoostingClassifier(
    min_samples_leaf=1,
    max_depth=2,
    learning_rate=1,
    max_iter=1
).fit(X_train_nn, Y_train)
y_pred_gdbt= gbdt.predict(X_test_nn)
accuracy_gbdt = accuracy_score(test_label, y_pred_gdbt)
print(f'GBDT Accuracy: {accuracy_gbdt:.4f}')
print(classification_report(test_label, y_pred_gdbt))
print(sns.heatmap(confusion_matrix(test_label, y_pred_gdbt), annot=True, fmt='d'))

rfc = RandomForestClassifier().fit(X_train_nn, Y_train)
y_pred_rfc = rfc.predict(X_test_nn)
accuracy_rfc = accuracy_score(test_label, y_pred_rfc)
print(f'RFC Accuracy: {accuracy_rfc:.4f}')
print(classification_report(test_label, y_pred_rfc))
print(sns.heatmap(confusion_matrix(test_label, y_pred_rfc), annot=True, fmt='d'))
```

## §3.6. Neural Networks

Multi-layer Perceptron (MLP) is a supervised learning algorithm that learns a function $f : R^m \to R^o$ by training on a dataset, where $m$ is the number of dimensions for input and $o$ is the number of dimensions for output. Given a set of features $X = x_1, x_2, ..., x_m$ and a target $y$, it can learn a non-linear function approximator for classification. It is different from logistic regression, in that between the input and the output layer, there can be one or more non-linear layers, called hidden layers. The following figure shows a one hidden layer MLP with scalar output.[2]

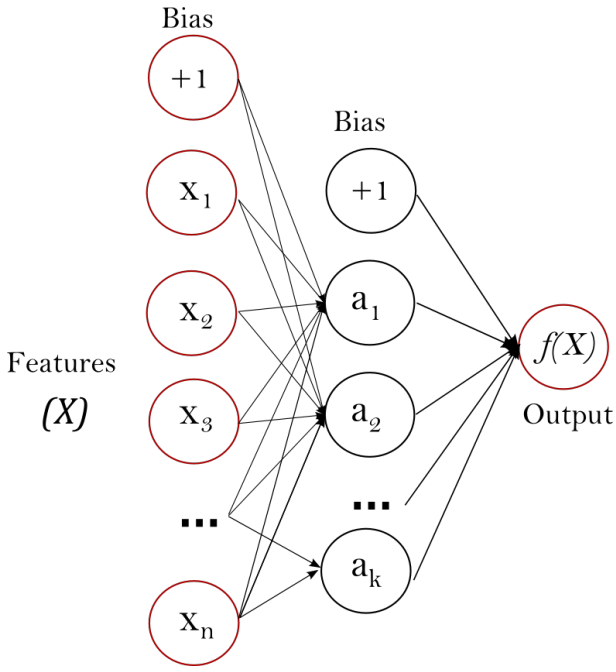


Figure 2: One hidden layer MLP

The leftest layer is known as the input layer which represents the input features. After a weighted linear summation and non-linear activation function, the neuron in the hidden layer will transform the inputs from previous layers. The output layer receives the values from the last hidden layer and transforms them into output values. Attributed to `sklearn`, we are not required to focus on how to construct the layers and activation function, we can utilize the neuron network by simply modifying on some parameters. The realization is as follows:

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(
    hidden_layer_sizes=(100,),
    activation='relu',
    solver='adam',
    alpha=0.0001,
    batch_size='auto',
    learning_rate='constant',
    learning_rate_init=0.001,
    power_t=0.5,
    max_iter=200,
    shuffle=True,
    random_state=None,
    tol=0.0001,
    verbose=False,
    warm_start=False,
```

```
    momentum=0.9,
    nesterovs_momentum=True,
    early_stopping=False,
    validation_fraction=0.1,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-08,
    n_iter_no_change=10,
    max_fun=15000
).fit(X_train_nn, Y_train)
y_pred_mlp = mlp.predict(X_test_nn)
accuracy_mlp = accuracy_score(test_label, y_pred_mlp)
print(f'MLP Accuracy: {accuracy_mlp:.4f}')
print(classification_report(test_label, y_pred_mlp))
print(sns.heatmap(confusion_matrix(test_label, y_pred_mlp), annot=True, fmt='d'))
```

# §4. Evaluation Method

As we notice in the Abstract that the data comes from the dataset Adult Census Income from Kaggle, we can use matching to find the real result of the `testdata.csv` and use this as reference to calculate our accuracy. The `testlabel.txt` is credited to one of my classmate Shengli Zhou, who kindly shares his method of testing the accuracy locally. Also, we try to use the method `classification report` and `confusion_matrix` to evaluate our classification results.

## §4.1. Classification Report

The classification report The classification report provides a detailed summary of the classification performance. It includes the following metrics for each class:

### §4.1.1. Precision

$$\text{Precision} = \frac{\text{True Positives(TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

### §4.1.2. Recall (Sensitivity or True Positive Rate)

$$\text{Recall} = \frac{\text{True Positives(TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

### §4.1.3. F1-Score

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

### §4.1.4. Support

The number of actual occurrences of the class in the dataset. This is simply the number of instances of each class.

## §4.2. Confusion Matrix

1. **True Positives (TP):** The number of correct predictions for class 1.

2. **False Positives (FP):** The number of incorrect predictions where the model predicted class 1 but the actual class was 0.
3. **True Negatives (TN):** The number of correct predictions for class 0.
4. **False Negatives (FN):** The number of incorrect predictions where the model predicted class 0 but the actual class was 1.

Using `sns` from `seaborn`, we can create a heatmap according to the confusion matrix as belows.

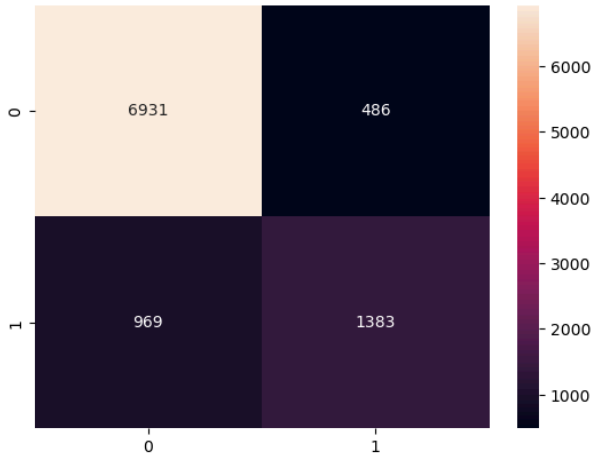


Figure 3: An example of the heat map

# §5. Performance Analysis

## §5.1. Decision Tree

In Decision Tree, we mainly focus on the `criterion` and `max_depth` two parameters. The results are as follows:

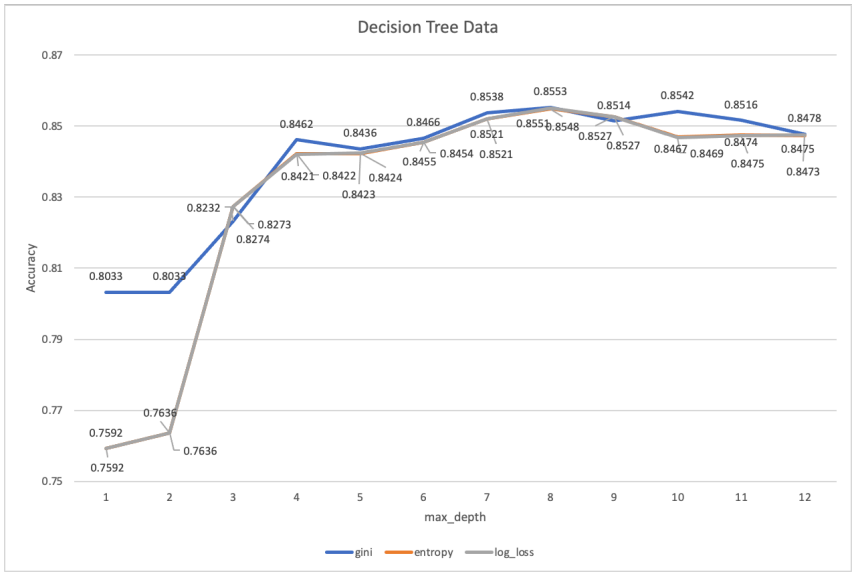


Figure 4: Experiments on Decision Tree

From the results we can see that the `gini` criterion is slightly better than the other two, which can reach its highest accuracy with a depth of 7. However, for all of them, the accuracy will go down due to overfitting when the max depth is increasing after 7. We have in total 14 attributes. By reaching its highest accuracy at the depth of 7, we can observe a very interesting phenom-

enon that it is not always the case that having a more detailed decision tree can contribute to a better classification prediction results.

```
Classification Report
              precision    recall  f1-score   support

           0       0.87      0.96      0.91      7417
           1       0.80      0.54      0.64      2352

    accuracy                           0.86      9769
   macro avg       0.83      0.75      0.77      9769
weighted avg       0.85      0.86      0.84      9769

Axes(0.125,0.11;0.62x0.77)
```

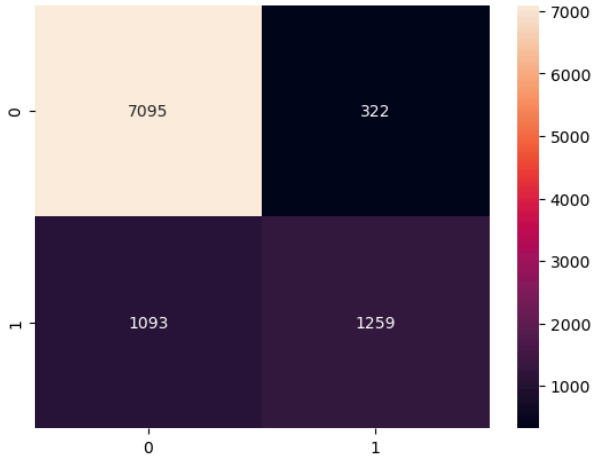


Figure 5: Heatmap for Decision Tree

From the classification report and the heat map for criterion `gini` and max depth `7`, we can see that our model play bad in distinguishing False Negatives and play well in distinguishing True Negatives. This indicates that our models have an inclination to mark the result as 0.

## §5.2. Naive Bayes

We haven't modified any basic parameter for each Bayes Model except for standardizing the inputs for Gaussian Naive Bayes and Bernoulli Naive Bayes. The inputs for Multinomial Naive Bayes and Complement Naive Bayes cannot be negative. After standardization, negative inputs will be created, therefore it is invalid to regard them as inputs for Multinomial Naive Bayes and Complement Naive Bayes.

```
Accuracy Report:
  Gaussian NB Accuracy: 0.7958
  Multinomial NB Accuracy: 0.7833
  Complement NB Accuracy: 0.7833
  Bernoulli NB Accuracy: 0.7892

  Gaussian NB Accuracy: 0.8280 (Standardized)
  Bernoulli NB Accuracy: 0.8033 (Standardized)
```

As we can see from the accuracy result, the standardization can help increase the accuracy of Gaussian Naive Bayes and Bernoulli Naive Bayes. However, contradicting with our assumption that Multinomial Naive Bayes and Complement Naive Bayes will perform better. Gaussian

Naive Bayes, which theoretically fits for continuous data, actually performs the best in such case.

## §5.3. Nearest Neighbors Classifiers

The parameters that can be changed in Nearest Neighbors Classifiers are different in the case of $k$-neighbors classifiers and radius neighbors classifiers. The former will be `weights`, which indicates the methods for calculating distance weights, and `n_neighbors`, which indicates the number of neighborhoods. The latter will be `weights` and `radius`, which indicates the radius of each neighborhood. We set the algorithm to `auto`, therefore it can choose the best algorithm for us based on our inputs.

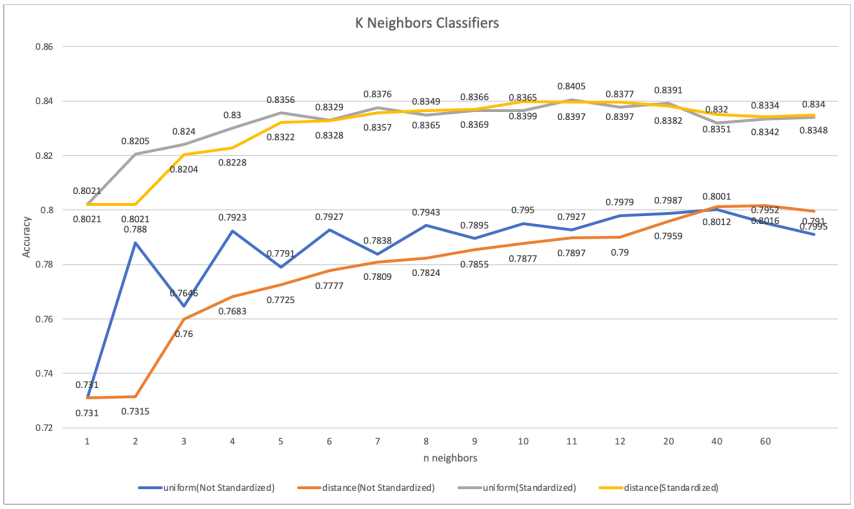


Figure 6: Experiments on K-Neighbors Classifiers

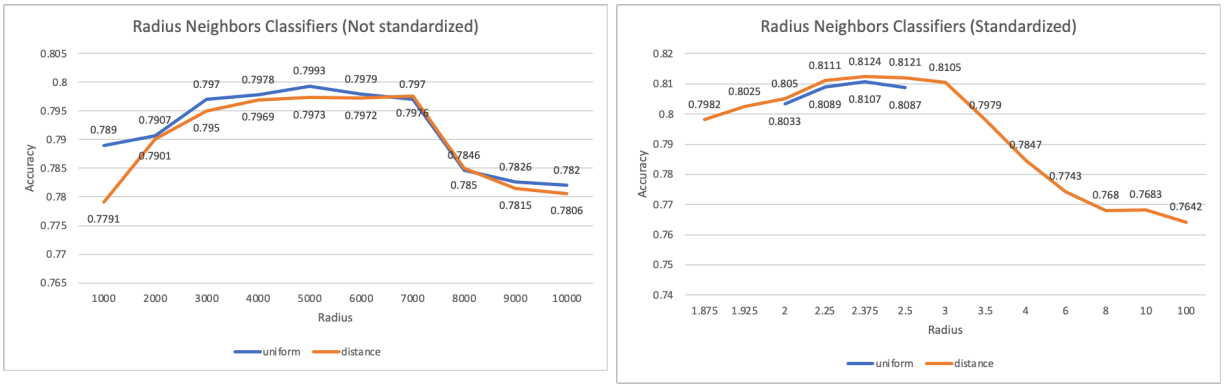


Figure 7: Experiments on Radius Neighbors Classifiers

We can see from the above experiments that using standardization will lead to an obvious increase in the result of Neighbors Classifiers, partially because this standardization will converge part of the data. However, we also notice that the radius used in non-standardization and standardization have a huge difference, because standardization will decrease the value we mapped to each attribute, making it more compact and making the numerical values lie between $-1$ and 1 as much as possible. In the K-neighbors classifiers and not standardized radius neighbors classifiers, using `uniform` weights exhibit a better performance while in standardized neighbors classifiers using `distance` weights exhibit a better accuracy. The trend also exhibit a mountain-like climb and drop due to underfitting and overfitting. However, in the not standardized `uniform` K-neighbors classifiers, the trend also exhibit an interesting zig-zag climb.

## §5.4. Support Vector Machines

The parameters that can be modified in support vector machine are `C` and `gamma`, which are usually used to control underfitting and overfitting. We use the `param_grid` to find out the best combination to be `C=1` and `gamma=0.1`. The according accuracy is 0.8475.

```
Classification Report
              precision    recall  f1-score   support

           0       0.87      0.94      0.90      7417
           1       0.75      0.55      0.63      2352

    accuracy                           0.85      9769
   macro avg       0.81      0.74      0.77      9769
weighted avg       0.84      0.85      0.84      9769

Axes(0.125,0.11;0.62x0.77)
```

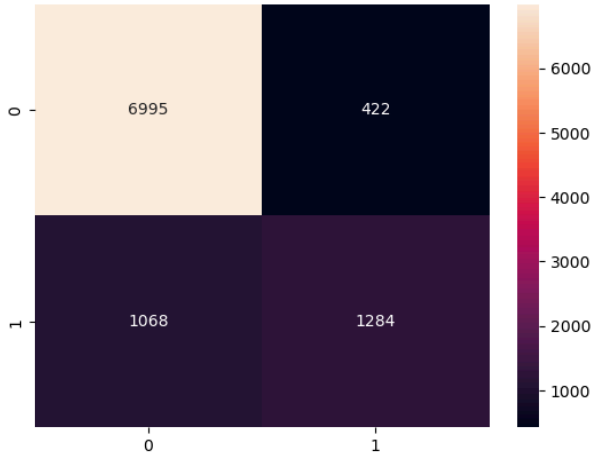


Figure 8: Heatmap for Support Vector Machine

We can see from the heatmap that still our model performs badly on the case of False Negatives and performs relatively well on the case fo True Negatives. This indicates that our trained data may have biases to some extent. Selecting the trained data may contribute to a better accuracy.

## §5.5. Ensemble Models

The parameter that can be modified in Ensemble Models is `max_depth`. Because the Gradient Boosting Classifier and the Random Forest Classifier are all based on decision tree as weak learners, therefore the choice of max depth should be similar to the decision tree process.

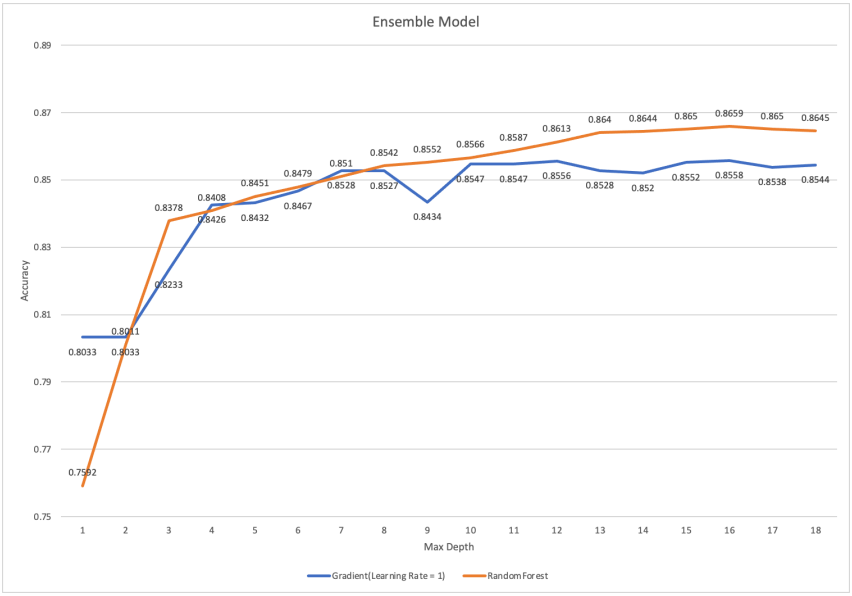


Figure 9: Experiments on Ensemble Models

We can see from the above results that the Random Forest Classifiers only exhibit a performance loss when the max depth reaches 16. The highest accuracy under the circumstance is 0.8659. The classification report and the heatmap at this point is as follows:

```
Classification Report
              precision    recall  f1-score   support

           0       0.88      0.95      0.91      7417
           1       0.79      0.60      0.68      2352

    accuracy                           0.87      9769
   macro avg       0.84      0.78      0.80      9769
weighted avg       0.86      0.87      0.86      9769

Axes(0.125,0.11;0.62x0.77)
```

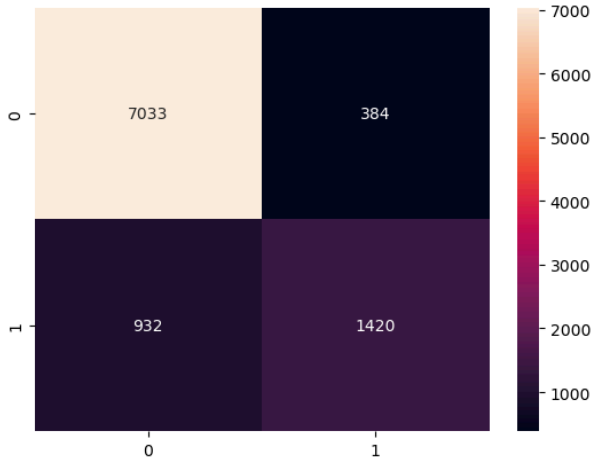


Figure 10: Heatmap for Ensemble Models

As we compare the heatmap with the above heatmaps, we can spot that the False Negatives instances and False Positive instances are decreasing, which show that the performance of Ran-

dom Forest can greatly reduce the false justification. However, it still cannot solve the bias for Negatives.

## §5.6. Neural Networks

In the experiments of Neural Networks, we try to find the best combination of the following parameters: `hidden_layer_size`, `activation`, `solver`, `alpha`. However, when we try to use the `param_size` method to find the best combination, the classifiers do not cease to produce warning of non-convergence. Hence, we can only turn to the more naive way to find the combination. After testing, we find that when we try to build a smaller neuron networks, the results incline to be better, probably because of our small training data size. The best result is produced under this parameter combination: (We also try different scalers, but the standard scaler is still the best.)

```
mlp = MLPClassifier(
    hidden_layer_sizes=(10,10,5),
    activation='tanh',
    solver='adam',
    alpha=0.001,
    batch_size='auto',
    learning_rate='adaptive',
    learning_rate_init=0.001,
    power_t=0.5,
    max_iter=200,
    shuffle=True,
    random_state=None,
    tol=0.0001,
    verbose=False,
    warm_start=False,
    momentum=0.9,
    nesterovs_momentum=True,
    early_stopping=False,
    validation_fraction=0.1,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-08,
    n_iter_no_change=10,
    max_fun=15000
).fit(X_train_nn, Y_train) # Using StandardScaler()
```

The best result is 0.8524, which is still inferior to the results produced by Random Forest Classifier.

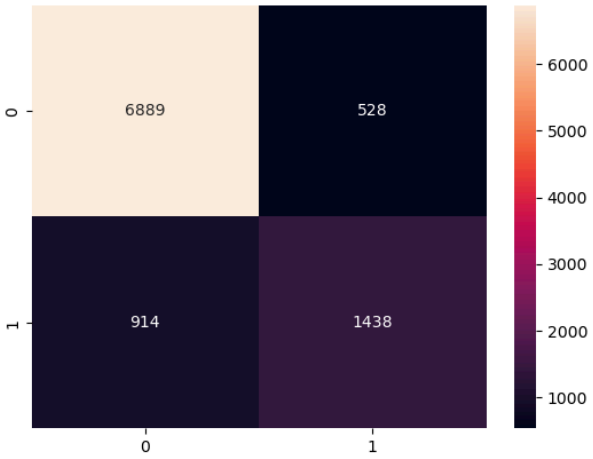


Figure 11: Heatmap for Neuron Networks

It is interesting to notice that in the heatmap of Neuron Networks, the number of False Negatives continue to decrease, while the number of False Positives tend to increase, leading to the worse performance. However, this shows that neuron networks may has a better capability coping with biased training data.

## §6. Conclusion and Further Plan

During this project, I have thoroughly conducted several hands-on practice on different classification methods. To some extent, it shatters my previous misconception of Neuron Networks being the most capable algorithm as Random Forest Classifier and Decision Tree Classifier can both out run Neuron Networks Classifiers. The best result we get in our experiment is from Random Forest Classifier, which is 0.8659.

However, there are still some pity in this experiment. First, I do learn things about how each classifier works, but I do not get the chance to realize it one by one (That will be a huge process...) Also, I have realized that there might be some data biased in our training data, but time does not permit me to further explore how to select out the biased term. Third, we have witnessed how powerful the scaling process is. There are actually all sorts of scalers provided in sklearn. Unfortunately, I fail to go over them, getting to know in what circumstances a scaler should be used. I think by improving from the above aspects, the results can be better.

At last, I want to express my gratitude to the teacher, Professor Yuan Bo, and all the teaching assistants for making this class enjoyable and enlightening. I do learn a lot about Artificial Intelligence through the course. Thank you for your devotion!

## §7. Reference

[1] 1.10. Decision Trees. (n.d.). Scikit-learn. https://scikit-learn.org/stable/modules/tree.html#tree-algorithms-id3-c4-5-c5-0-and-cart

[2] 1.17. Neural network models (supervised). (n.d.). Scikit-learn. https://scikit-learn.org/stable/modules/neural_networks_supervised.html