Data Mining

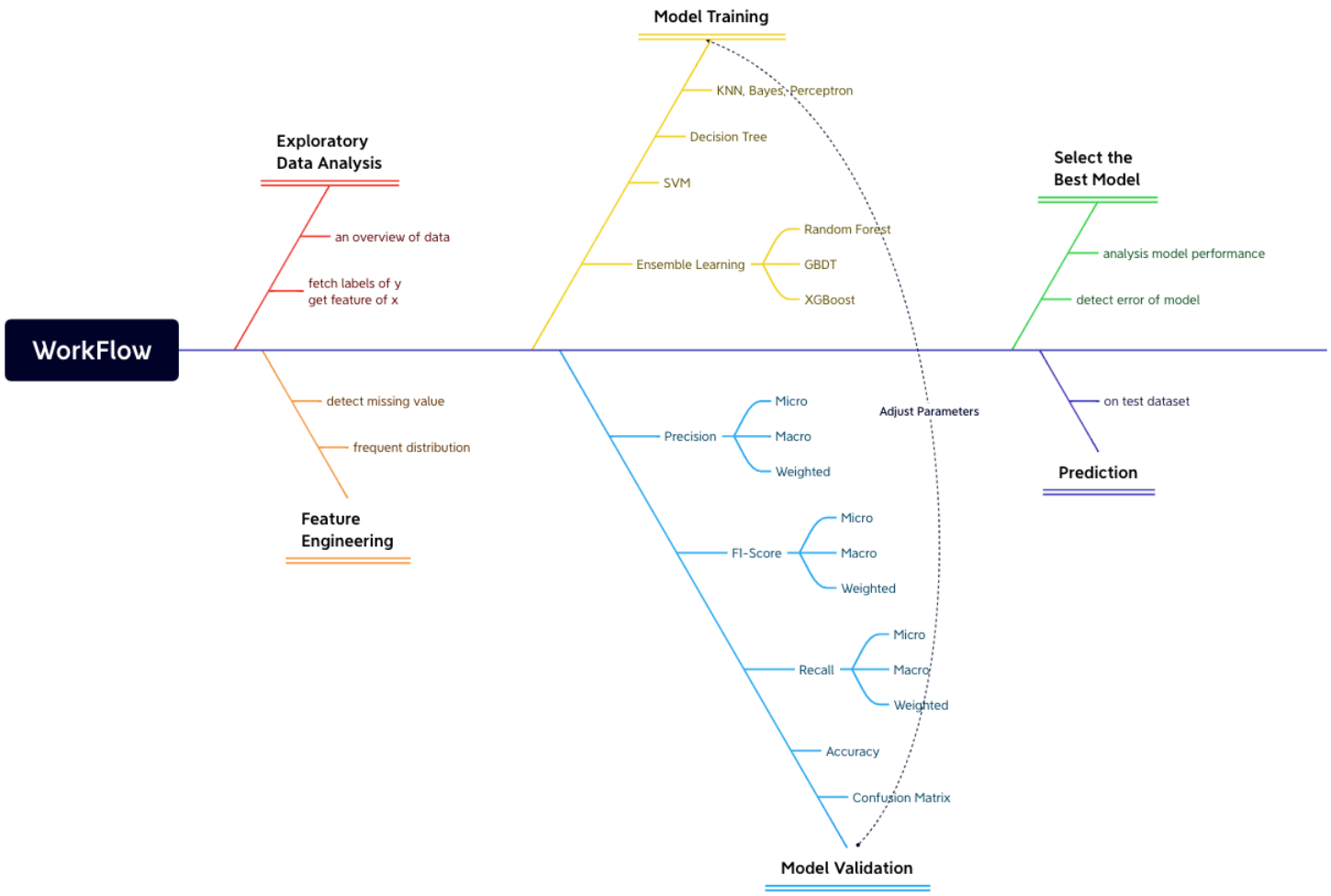
Individual Project Report

Name: Ruoxin WANG

Student ID: 2030026150

Section: Data Mining (1002)

1 The Workflow



1.1 Exploratory Data Analysis

Explore the basic characteristics of the dataset.

The number of items in “training.csv” is 1330.

And there are 6 features as x, and all of them are categorical. And column “evaluation” as y, has three possible values.

All the features have no missing value.

```
Shape of data:  (1330, 7)
```

	buying	maint	doors	persons	lug_boot	safety	evaluation
0	low	vhigh	5more	more	small	low	unacc
1	high	high	2	2	big	med	unacc
2	low	vhigh	3	2	med	med	unacc
3	vhigh	low	5more	2	big	med	unacc
4	vhigh	vhigh	4	2	big	med	unacc

	count	unique	top	freq
buying	1330	4	high	350
maint	1330	4	high	342
doors	1330	4	3	346
persons	1330	3	2	453
lug_boot	1330	3	med	456
safety	1330	3	low	463

```
RangeIndex: 1330 entries, 0 to 1329
Data columns (total 6 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   buying    1330 non-null   object
 1   maint     1330 non-null   object
 2   doors     1330 non-null   object
 3   persons   1330 non-null   object
 4   lug_boot  1330 non-null   object
 5   safety    1330 non-null   object
dtypes: object(6)
```

1.2 Feature Engineering

Firstly, we found that there is no missing value for all features.

Secondly, find out the distribution of each feature, and we can conclude that all features distribute well-proportioned.

```
high     350
vhigh    340
low      320
med      320
Name: buying, dtype: int64
high     342
vhigh    341
low      328
med      319
Name: maint, dtype: int64
3        346
2        338
4        325
5more    321
Name: doors, dtype: int64
2       453
more    441
4       436
Name: persons, dtype: int64
med      456
small    450
big      424
Name: lug_boot, dtype: int64
low     463
med     458
high    409
Name: safety, dtype: int64
```

For y, the target attributes, 968 items have ‘unacc’ value, 307 items have ‘acc’ value and 55 items have ’good’ value.

Next, test the independence of each x with y; we have the following table:

Features	buying	maint	doors	persons	lug_boot	safety
p-value	92.59	6.19	3.04	5.16	3.88	6.59
Chi-square	8.78e-18	0.72	0.96	0.52	0.69	0.36

We can conclude that the feature ‘buying’ depends on y, and other features are all independent of y.

1.3 Model Training

Here I implemented nine models including KNN, Bayes, Perceptron, which implemented myself, and Decision tree and SVM using the sklearn package. Apart from this, I used some assembling learning models like GBDT, Random Forest, and XGBoost to fit models.

All models I encapsulated into class it is easier to use when calling them to fit the model and predict the result.

1.4 Model Validation

I defined a class called Score to calculate each model's performance, including confusion matrix, accuracy, recall, F1-score, and precision. I also draw a heatmap of the confusion matrix to analyze it more intuitionistically.

1.5 Select Best Model

By comparing all criteria of all models, we find that SVM-OVO performs best on prediction, so I select SVM with one VS one method as the final model.

2 The Model Adopted

2.1 KNN Model

This model requires inputting training data after encoding data and inputting valid data one by one. And user can set k by themselves; when testing data, I used 5, which means finding the five closest points to the target point.

In this model, we first calculate the distance between items and the target point using Euclidean distance. And according to the item's label, assign a label which appears the most frequently to target as a result. When we want to use this function, we must encode the categorical data; here I use the OneHotEncoding method.

2.2 Bayes Model

This model requires inputting the whole dataset.

It first spilled data into training and validating set, then calculate prior probability and condition probability, respectively, according to the formula:

$$P(A|B) = \frac{P(B|A) \times P(B)}{P(B)}$$

$$P(C_i)P(x|C_i) = P(C_i) \times \prod_{i=1}^{n} P(X_i|C_i)$$

And using two of them multiplied together as posterior probability and using the label which has the largest probability as a result.

2.3 Perceptron Model

This model requires a training set, a testing set as input, and above two datasets must be encoded by OneHotEncoder. And we can also set the learning rate and the number of iterations.

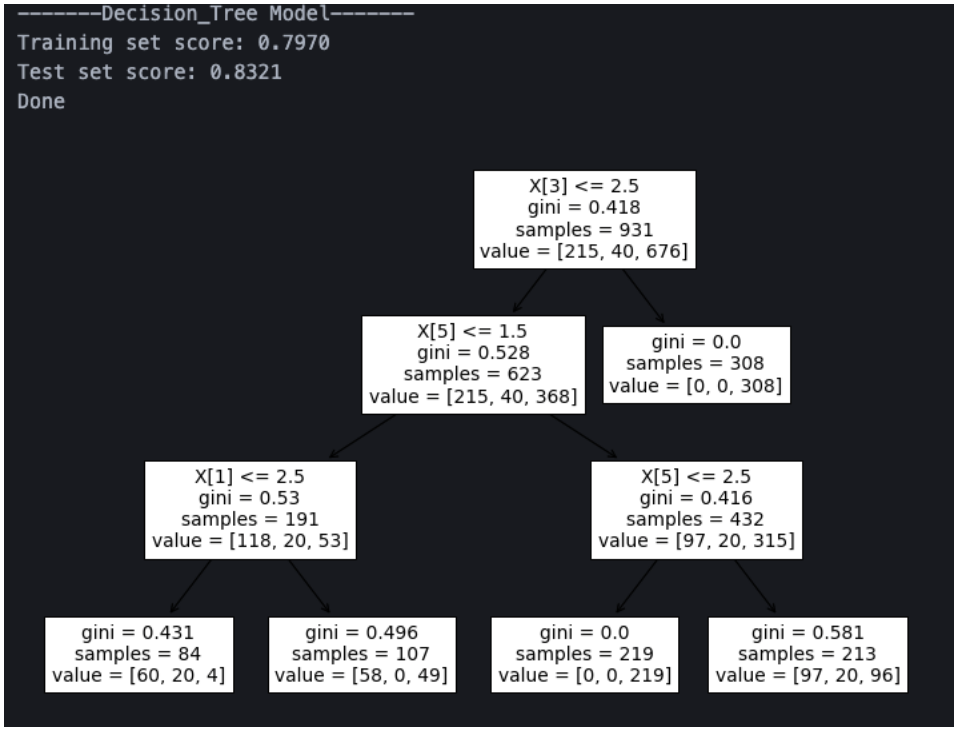
I use nested perceptron in this model, first dividing them into ‘unacc’ and “‘good’ or ‘acc.’” Then further check whether it is ‘good’ or ‘acc.’ For data preprocessing, I use 0 and 1 to encode the data by using two maps. And using two weight and bias variables to store two parameters of two Perceptron Models, respectively.

After training, it uses nested if-statement to predict the result.

2.4 Decision Tree Model

This model requires a training set and testing set as input. And this model does not need to be encoded, so it just spilled data before training.

In this model, I used Gini Index to judge which feature should be selected to construct the tree. And after training, I use the score function in sklearn to check whether the model encounters an overfitting problem. Finally, use the Graphviz package to visualize the decision tree and return the predicted result by using the model which has been trained just now.



2.5 GBDT Model

Gradient Boosting Decision Tree Model is an ensemble model; its idea is to use a set of CART decision trees, and each tree learns the residual of the sum of the conclusion of all previous trees. And all trees are related iteratively. Consequently, the final gradient boosting tree is used to predict the test set. And here, I set hyperparameters ‘n_estimators’ and ‘max_features,’ the same as the Random Forest model shown behind this. To determine these hyperparameters, I use the GridSearch method.

2.6 Logistic Regression Model

This model requires data as the primary input, and sets some hyperparameters like ‘solver,’ ‘max_iter,’ ‘multi_class’ and ‘class_weight’ as input.

Before training the model, we need to encode data using OneHotEncoder. After that, split data into training data and testing data and passing parameters to the LosigticRegression function in the sklearn package to get the predicted result.

2.7 Random Forest Model

The random forest model is an ensemble model; its basic idea is to form some decision trees which are independent of each other to predict results.
This model requires data as input and uses the spilt_data method to divide it into a training set and a valid set. Here, we use the grid_search method to find the best hyperparameter to fit the model. So we set 'n_estimators = 200' and 'max_features = 5' and use this hyperparameter in the GBDT model.

2.8 SVM Model

This model requires data as input, and we can set the strategy of the fitting model as either 'OVO' or "OVR." This model first encodes data by using OneHotEncoder and fits two strategy, respectively. And I set hyperparameter 'kernel = 'poly''. It means the method will use a polynomial function as a kernel function; it is better than the 'linear' function I used to utilize.

2.9 XGBoost Model

XGBoost Model is an ensemble model using a set of the decision tree. And the tree in it is related iteratively. The core process of it is to sort features to determine the best point to divide. And it supports cross-validation inside itself, so it quickly gets the best iteration time, which we do not need to set.

This model requires data as input and uses a function in sklearn. And we first need to let data be encoded; here, we use OrdinalEncoder to encode x and y. When fit model, I set hyperparameter ‘eval_metric = ‘mlogloss’’, and use ‘n_estimators = 200’ and ‘max_features = 5’, which learned in Random Forest Model.

3 Experimental Results

3.1 KNN

	Accuracy	Recall	F1-Score	Precision
Micro	0.9173	0.9173	0.9173	0.9172
Macro		0.7565	0.7772	0.8068
Weighted		0.9173	0.9153	0.9144

3.2 Bayes

	Accuracy	Recall	F1-Score	Precision
Micro	0.8697	0.8697	0.8697	0.8697
Macro		0.6392	0.6664	0.7178
Weighted		0.8697	0.8631	0.8601

3.3 Perceptron

	Accuracy	Recall	F1-Score	Precision
Micro	0.9323	0.9323	0.9323	0.9323
Macro		0.9059	0.8538	0.8261
Weighted		0.9323	0.9334	0.9381

3.4 Decision Tree

	Accuracy	Recall	F1-Score	Precision
Micro	0.8321	0.8321	0.8321	0.8321
Macro		0.6073	0.5451	0.5262
Weighted		0.8321	0.8293	0.8652

3.5 GBDT

	Accuracy	Recall	F1-Score	Precision
Micro	0.9774	0.9774	0.9774	0.9774
Macro		0.9476	0.9393	0.9320
Weighted		0.9774	0.9778	0.9787

3.6 Logistic Regression

	Accuracy	Recall	F1-Score	Precision
Micro	0.9148	0.9148	0.9148	0.9148
Macro		0.8670	0.8290	0.8019
Weighted		0.9148	0.9187	0.9311

3.7 Random Forest

	Accuracy	Recall	F1-Score	Precision
Micro	0.9674	0.9674	0.9674	0.9674
Macro		0.9380	0.9213	0.9060
Weighted		0.9674	0.9680	0.9694

3.8 SVM

3.8.1 OVO

	Accuracy	Recall	F1-Score	Precision
Micro	0.9824	0.9824	0.9824	0.9824
Macro		0.9920	0.9663	0.9436
Weighted		0.9824	0.9827	0.9837

3.8.2 OVR

	Accuracy	Recall	F1-Score	Precision
Micro	0.9674	0.9674	0.9674	0.9674
Macro		0.9194	0.9091	0.9003
Weighted		0.9674	0.9679	0.9697

3.9 XGBoost

	Accuracy	Recall	F1-Score	Precision
Micro	0.9724	0.9724	0.9724	0.9724
Macro		0.9428	0.9260	0.9106
Weighted		0.9724	0.9729	0.9743

According to table for each model, we can conclude that SVM-OVO perform best, and we further use k-fold cross validation for it, we can find that:

	Mode	**Accuracy**	**Recall**	**F1-Score**	**Precision**
1-Fold	**Micro**	0.9925	0.9925	0.9925	0.9925
	Macro		0.9897	0.9930	0.9965
	Weighted		0.9925	0.9924	0.9926
2-Fold	**Micro**	0.9925	0.9925	0.9925	0.9925
	Macro		0.9881	0.9681	0.9524
	Weighted		0.9925	0.9927	0.9936
3-Fold	**Micro**	0.9887	0.9887	0.9887	0.9887
	Macro		0.9469	0.9542	0.9626
	Weighted		0.9887	0.9885	0.9886
4-Fold	**Micro**	0.9962	0.9962	0.9962	0.9962
	Macro		0.9933	0.9791	0.9667

	Weighted		0.9962	0.9963	0.9966
5-Fold	**Micro**	0.9925	0.9925	0.9925	0.9925
	Macro		0.9903	0.9618	0.9394
	Weighted		0.9925	0.9928	0.9938
Average	**Micro**	0.9925	0.9925	0.9925	0.9925
	Macro		0.9816	0.9712	0.9645
	Weighted		0.9925	0.9925	0.9930

SVM with OVO mode performs very well in K-Fold cross-validation.

4 Result Analysis

4.1 SVM-OVO Model achieves the BEST performance on this dataset.

1. SVM is a convex optimization problem, in essence. So, if the additional sample has no use for restriction, it will not affect the final result. That's why SVM is suitable for small-scale data.

2. SVM is good at solving linear inseparability problems by using kernel function and slack variables.

3. The classifier is only determined by support vectors, so the SVM model can efficiently avoid overfitting problems.

4. After one-hot encoding, the dimension of the dataset has reached 21 dimensions, while SVM is very concise and only uses support vectors. Therefore, it is suitable for processing high-dimensional data.

4.2 Error Analysis

1. Decision Tree Model

1) According to the result of feature engineering, the distribution of y' label is imbalanced. And decision trees prefer classes with more samples, so the class with more samples tends to be correctly classified, while classes with fewer samples will be sacrificed.

2) We construct a decision tree recursively, so it will generate exceed vertex for the final model, leading to an overfitting problem. Recall that we calculate the score of overfitting when fit model; it could be better.

3) Sample in the dataset has few representative points, so some data classes need to fit better. According to the confusion matrix of the decision tree, we can generate the same conclusion.

2. Bayes

1) One basic assumption of Naive Bayes is that all features mutually independent. But according to previous feature engineering, we can find that, except the 'buying' feature, all other features have a relatively high correlation with y. So there exists some error in assumption, and the model does not fit well.

2) We use likelihood and prior probability to predict posterior to determine class. So, it exists some errors when doing determination using this method.

3. Logistic Regression

1) Logistic Regression model is used to solve linear problems, so it does not perform well when encountering nonlinear problems. And it is a waste of time to convert the nonlinear problem into linear problems.

2) According to the result of feature engineering, the distribution of y' label is imbalanced. And Logistic Regression cannot deal with imbalanced data well.

3) The model is too simple to simulate the proper distribution of the data. It means Logistic Regression cannot capture complex relations in data.

4) It is sensitive to the multicollinear problem, just like the Linear Regression model.