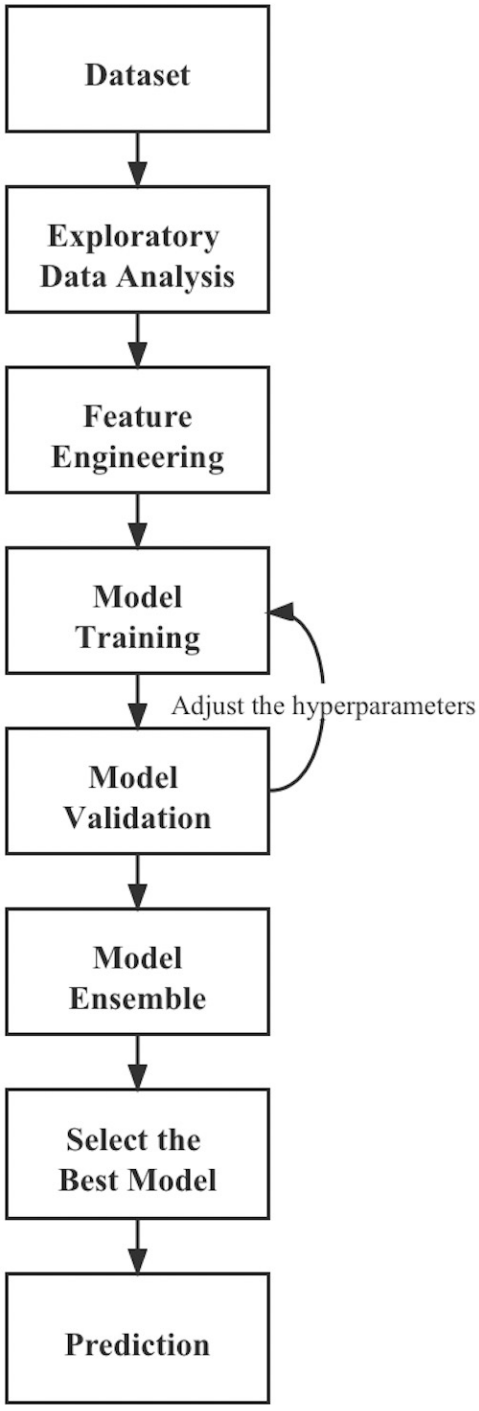
Individual Project Report

Data Mining

杨皓言 1930026145

1. Workflow



- **Exploratory Data Analysis**

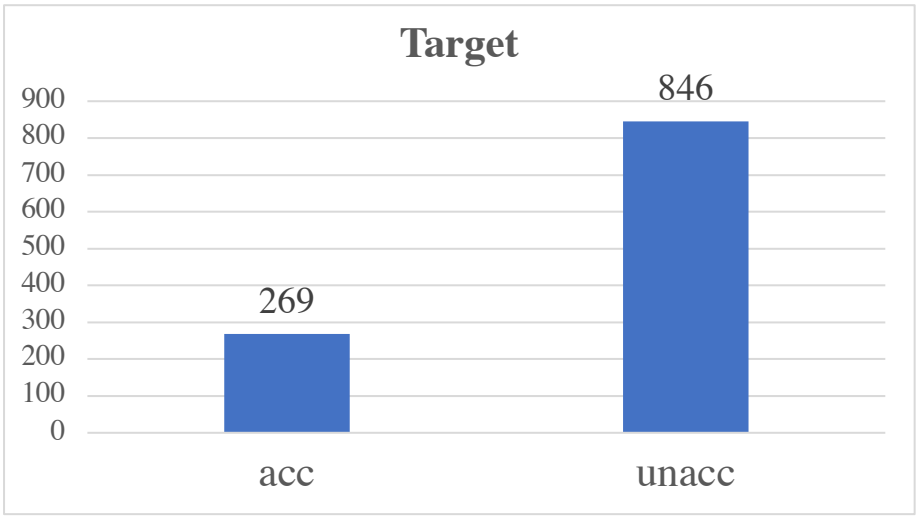
Do the most basic exploration of the data, including the number of the data, the type of data, whether there are missing values and outliers, the distribution of the data.

There are 6 features and 1 column of target values. And for every feature, there are 1115 data, no null values, and no outliers. All data types are discrete.

	buying	maint	doors	persons	lug_boot	safety	evaluation
count	1115	1115	1115	1115	1115	1115	1115
unique	4	4	4	3	3	3	2
top	high	vhigh	2	2	small	low	unacc
freq	308	301	298	387	382	408	846

```
Data columns (total 7 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   buying       1115 non-null   object
 1   maint        1115 non-null   object
 2   doors        1115 non-null   object
 3   persons      1115 non-null   object
 4   lug_boot     1115 non-null   object
 5   safety       1115 non-null   object
 6   evaluation   1115 non-null   object
dtypes: object(7)
```

For the target values, there are 2 types —— ‘acc’and ‘unacc’. And there are 269 acc values and 846 unacc values.



- **Future Engineering**

Analyze the features of the data, including data preprocessing, analyze the correlation between the features and the target, whether it is necessary to reduce the dimensionality of the features

Because all data is discrete, we need to encode it. I used two methods, one-hot encoding, and label encoding, each of which is suitable for different models.
At the same time, to fit and validate the model, I divided the given training set into a training set and a validation set. The size ratio of the training set and the validation set is 0.25:0.75.

Next, I analyzed the independence between each feature and the target through the chi-square test. The result is

Features	buying	maint	doors	persons	lug_boot	safety
p-value	0.00004	0.0017	0.3839	$5.79*10^{-42}$	0.0066	$6.18*10^{-46}$

So, we know except the feature 'doors', all features are dependent on the target. But because there are not many features, in order to get more information from features, I do not delete this feature 'doors'.

Lastly, because the type of data is discrete and there are not too many features, I decide not to perform feature reduction and standardization on it after encoding.

- **Model Training**

I implement 4 models by myself: KNN, Bayes, Perceptron and Decision Tree
And I adopt 4 models to the task from the scikit-learn: Logistic Regression, SVM, Random Forest and Gradient Boosting Decision Tree.
And I use the train set divided before to fit these models

- **Model Validation**

In order to get better performance of models, I use the validation set divided before to adjust the hyperparameters of every model, and I also use some techniques like k-fold validation to avoid the unstable and poor model performance due to accidental problem from division of training set and test set, and like grid research to find the best hyperparameters combination automatically.

- **Model Ensemble**

Since each model has its own shortcomings, model ensemble can avoid this situation to a certain extent and optimize the performance of the model. I used SVM, Random Forest, Gradient Boosting Decision Tree, Decision Tree and Bayes 5 signal models for ensemble. Finally, vote based on the results calculated by multiple models to determine the target value.

- **Select the Best Model and Prediction**

I use the **SVM model** as the final model to predict the test set and give the target result.

2. Adopted Models

Firstly, I implement 4 models by myself: KNN, Bayes, Perceptron and Decision Tree

(1) KNN Model: it needs a train set to fit the model, and then put the test set into the model to get the predictions. Besides, it also has a hyperparameter k, which means select k closest points between the test point and use the target what majority these k points are as the result. I use the Euclidean distance $d = \sqrt{(x_i - x_j)^2}$ to calculate the distance between the test point and all train points and use the target of majority 6 closest points are as the prediction.

(2) Bayes Model: it needs a train set to fit the model and you can put the test set into the model to get the predictions. There are not hyperparameters in the model. And the base of the model is bayes rule $\mathrm{P(A|B)} = \frac{\mathrm{P(B|A)} \times \mathrm{P(B)}}{\mathrm{P(B)}}$, and we calculate the probability that $\mathrm{P(C_i)P(x|C_i)} = \mathrm{P(C_i)} \times \prod_{i=1}^{n} \mathrm{P(x_i|C_i)}$ to decide which target it is more likely to belong to.
According to the formula, my model calculates the prior probability firstly and then calculates the conditional probability, finally multiple them to get the probability of different targets, select the target whose probability are maximum. Moreover, I use the Laplacian correction to avoid that there is no tuple of $\mathrm{C_i}$ having $\mathrm{A} = \mathrm{x_k}$, increasing the universality of the model.

(3) Perceptron Model: it needs a train set to fit the model and you can put the test set into the model to get the predictions. There are two hyperparameters in the model, "iteration times" and "learning rate". Iteration times determines how many times you update the weights and "learning rate" determines whether your model can converge and how long it takes to converge. In my model, I set the iteration times to 20 and learning rate to 0.01. In addition, the model also needs an activation function. I use the step function $f(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$. At the beginning, the model has a random weight from the range -1 to 1 and bias is 0. It will update the weight and bias according to the equation $\mathrm{w_j} = \mathrm{w_j} + \alpha(\mathrm{T} - \mathrm{O})\mathrm{I_j} + \varepsilon$ and $\varepsilon = \varepsilon + \alpha \mathrm{I_j}$ (α is the learning rate and ε is the bias) after each iteration. If it has converged, my model will terminate the iteration early. After finishing the iterations, it gets the final weight and I use the weight through the step function to get the final target.

(4) Decision Tree: it needs a train set to construct a decision tree and you can put the test set into the model to get the predictions. There are not hyperparameters in the model. And I use the Gini coefficient to judge which feature should be selected in this turn to construct the tree. In addition, there are two base cases to terminate to construct the tree, one is there are only one kind of target in the dataset, and another

is there are only one feature in the dataset. At the same time, to avoid overfitting, I also set the extra termination condition that if the Gini coefficient of a feature is larger than 0.4, we also stop branching.

Secondly, I adopt 4 models to the task from the scikit-learn: Logistic Regression, SVM, Random Forest, and Gradient Boosting Decision Tree

(5) **Logistic Regression:** I import the LogisticRegression from the library sklearn.linear_model. The principle is to use a logistic function to map the result of linear regression $(-\infty, \infty)$ to (0, 1) and then do classification.

(6) **SVM:** I import the SVC from the library sklearn.svm. And I set two hyperparameters, kernel = 'poly' and C=1. "kernel" specifies the kernel function, which is a method used to transform nonlinear problems into linear problems. "C"is used to control the penalty coefficient of the loss function. The basic idea of SVM is to solve the separation hyperplane that can correctly divide the training data set and have the largest geometric interval. Then use the hyperplane to predict the target of test set.

(7) **Random Forest:** I import the RandomForestClassifier from the library sklearn.ensemble. Actually, it is an ensemble model of decision tree, which means many decision trees form a random forest. The principle is to take the target what majority decision trees are as the final prediction result. I set two hyperparameters, "n_estimators = 200" and "max_features = 6" through grid research. "n_estimator" determines how many trees you use to form a random forest and "max_features" determines the maximum depth of each decision tree.

(8) **Gradient Boosting Decision Tree:** I import the GradientBoostingClassfier from the library sklean.ensemble. It is also an ensemble model. The core is that each tree learns the residual of the sum of the conclusions of all previous trees. Compared with each tree in the random forest is independent, in the gradient boosting tree, each tree is related, and the final gradient boosting tree is used to predict the test set. I set two same hyperparameters in random forest, "n_estimators = 200" and "max_features = 6".

Thirdly, I use SVM, Random Forest, Gradient Boosting Decision Tree, Decision Tree and Bayes 5 signal models for ensemble. Vote based on the results calculated by multiple models to determine the target value.

Finally, I find SVM model has the best performance and in order to validate the stability of SVM model, I also use K-folds to test it and I find that every time the performance is excellent compared to the ensemble model. Therefore, **I use the SVM model as my final model.**

3. Experimental Results

Implemented models by myself

(1) KNN

	accuracy	precise	recall	f1_score
Use acc as positive	0.9642	0.9655	0.8750	0.9180
Use unacc as positive	0.9642	0.9638	0.9907	0.9971

(2) Bayes

	accuracy	precise	recall	f1_score
Use acc as positive	0.9283	0.8413	0.8413	0.8413
Use unacc as positive	0.9283	0.9537	0.9537	0.9537

(3) Perceptron

	accuracy	precise	recall	f1_score
Use acc as positive	0.9032	0.7750	0.8732	0.8212
Use unacc as positive	0.9032	0.9548	0.9135	0.9337

(4) Decision Tree

	accuracy	precise	recall	f1_score
Use acc as positive	0.9032	0.7234	0.9855	0.8344
Use unacc as positive	0.9032	0.9946	0.8762	0.9316

Use the existing library

(5) Logistic Regression

	accuracy	precise	recall	f1_score
Use acc as positive	0.9642	0.8733	0.9841	0.9254
Use unacc as positive	0.9642	0.9952	0.9583	0.9764

(6) Random Forest

	accuracy	precise	recall	f1_score
Use acc as positive	0.9892	0.9545	1.0	0.9767
Use unacc as positive	0.9892	1.0	0.9861	0.9930

(7) Gradient Boosting Decision Tree

	accuracy	precise	recall	f1_score
Use acc as positive	0.9892	0.9524	1.0	0.9756
Use unacc as positive	0.9892	1.0	0.9863	0.9931

(8) Ensemble Model

	accuracy	precise	recall	f1_score
Use acc as positive	0.9928	0.9859	0.9859	0.9859
Use unacc as positive	0.9928	0.9952	0.9952	0.9952

(9) SVM (the final model and do 5-fold validation)

		accuracy	precise	recall	f1_score
1st fold	Use acc as positive	1.0	1.0	1.0	1.0
	Use unacc as positive	1.0	1.0	1.0	1.0
2nd fold	Use acc as positive	0.9955	0.98	1.0	0.9899
	Use unacc as positive	0.9955	1.0	0.9943	0.9971
3rd fold	Use acc as positive	0.9910	0.9643	1.0	0.9818
	Use unacc as positive	0.9910	1.0	0.9982	0.9940
4th fold	Use acc as positive	1.0	1.0	1.0	1.0
	Use unacc as positive	1.0	1.0	1.0	1.0
5th fold	Use acc as positive	0.9955	1.0	0.9818	0.9908
	Use unacc as positive	0.9955	0.9941	1.0	0.9970
Average	Use acc as positive	**0.9964**	**0.9889**	**0.9964**	**0.9925**
	Use unacc as positive	**0.9966**	**1.0**	**0.9981**	**0.9978**

4. Result Analysis

(a) **SVM model** has the best performance on this dataset.
Reasons:

(1) The number of samples required by SVM is relatively small, and it often has good performance on this not very large dataset.

(2) SVM is good at coping with the linear inseparability of sample data, which is mainly realized by kernel function and slack variables. This is why the Logistic Regression model does not perform well in many models that call the library, because it performs linear regression on the data.

(3) The classification idea of SVM is very simple, which is to maximize the interval between the sample and the decision surface. Because the classifier generated by SVM is very concise and only uses support vectors, it is more suitable for processing high-dimensional data. After one-hot encoding, the dimensionality of the data has reached 22 dimensions, and the SVM has advantages in high-dimensional datasets.

(4) Since the classifier is only determined by the support vector, SVM can also effectively avoid overfitting.

(b) On the dataset, for the models which implemented by myself, **Perceptron** does not perform well.
Reasons:

(1) The principle of the perceptron is to iterate over the data to update the weights. Therefore, the size of the data set is closely related to the performance of the perceptron. This dataset is not large enough is one of the reasons why the

perceptron performance is not very good.

(2) The perceptron we use has only one layer, which is not enough for him to learn enough useful information

For the models I call them from existing library, **Logistic Regression** dose not perform well.

Reasons:

(1) Can't use logistic regression to solve nonlinear problems

(2) The ratio of the two types of targets is about 1:3, it is difficult to deal with the problem of data imbalance

(3) The form is simple, and it is difficult to simulate the true distribution of the data. At the same time, the data are all discrete, so the effect after fitting with regression is not good.