Traditional Chinese Medicine(TCM) Prescription Modeling

(1) Introduction

Insomnia is a common sleep disorder that affects many people around the world. Traditional Chinese Medicine (TCM) is an alternative treatment that can be used to treat insomnia. We can search some information from the Internet, and it becomes easier to obtain various cases about Chinese medicine and insomnia. In this case, we'll explore some of the sources that provide information about TCM and insomnia. We'll also discuss how to classify and organize data. In addition, we also collected various TCM prescriptions for treating different types of insomnia. Our goal is to use machine learning to automatically recommend TCM prescriptions based on patient profiles.

1. **Information Sources:**

We searched for some relevant information about Chinese medicine and insomnia from the online platform "Jing Fang Pai". The site has several articles and case studies related to insomnia. In addition, various academic journals and research papers can be found on the Internet. These papers provide valuable insights into the mechanism of action of TCM in treating insomnia.

2. **Classification and Organization:**

Once we have collected the data, the next step is to classify and organize it. One approach is to categorize the data by age and specific symptoms. For example, we can divide the data into different age groups, such as young (18-35), middle-aged (36-60), and old (61+). We can also categorize the data based on specific symptoms such as trouble falling asleep, frequent night wakings, early awakenings, headaches, dizziness, tongue coating and other related symptoms.

3. **Purpose and Goal:**

Our purpose is to develop a predictive model that can recommend specific TCM prescriptions based on individual patient profiles. This project aims to utilize ML to provide

personalized and effective TCM treatments for insomnia, thereby enhancing the patient's quality of life.

Our goal is to use machine learning to recommend TCM prescriptions for people suffering from insomnia. By analyzing the data we collected, we were able to identify patterns and associations between patient profiles and TCM prescriptions for successful treatment of insomnia. This information can then be used to develop predictive models that can recommend specific TCM prescriptions based on a patient's profile. Ultimately, this could improve the effectiveness of TCM treatments and provide patients with personalized and effective insomnia treatment.

(2) Data Processing

1. Data Collection:

The study embarked on processing data from 200 medical cases, particularly focusing on the application of TCM in the treatment of insomnia. The data was sourced from Jing Fang Pai, a comprehensive online repository that boasts a wealth of medical cases centered on TCM. The dataset was extensive and incorporated 28 features including **Gender, Age, Dysmenorrhea, Menstrual Description, Medical History, Difficulty Falling Asleep, Waking Up at Night, Headache, Facial Color, Pulse Condition, Pulse Position, Pulse Width, Fluency, Tension, Tongue Diagnosis, Tongue Coating, Tongue Coating Color, Tongue Coating Thickness, Appetite, Dry Mouth, Night Urination, Constipation, Acne, Easily Irritable, Urination ,Urination Status, Sweating, Dizziness, Palpitations** *and* **Chest Tightness.**

2. Initial Processing:

The raw data required substantial processing to transform it into a format that would be compatible with machine learning (ML) algorithms. The procedure was carried out meticulously to ensure the final dataset would be conducive to seamless integration with subsequent ML techniques.

The first step in the data processing endeavor involved replacing the original Chinese data for **Gender** and **Medical History** with 'F' for female, 'M' for male, and the duration of the medical history in months respectively. This step was fundamental in eliminating any potential language barrier that could hinder the effectiveness of the ML algorithms.

The binary features, such as **Difficulty Falling Asleep, Waking Up at Night,** and **Headache,** were also transformed to 'Y' for presence and 'N' for absence of the respective conditions. This binary format is widely recognized in machine learning practices, enabling the algorithms to understand and interpret these features more effectively.

Facial Color, a categorical feature, was transformed into alphabetic data, with a, b, c, d, and e representing different facial colors and conditions. This transformation was vital in converting qualitative information into a quantifiable format that could be processed by ML algorithms. **Pulse Position, Pulse Width, Fluency,** and **Tension**, all features related to pulse diagnostics, were converted to alphabetic data. Each state of these pulse attributes was represented by a distinct alphabet, which essentially translated the intricate art of pulse diagnostics in TCM into a language that ML algorithms could decipher.

Tongue Diagnosis, Tongue Coating Color, and **Tongue Coating Thickness,** which are important indicators in TCM, were also converted into alphabetic data. Each specific condition or appearance was assigned a unique alphabet. This transformation was crucial in preserving the diagnostic essence of TCM while making it compatible with ML algorithms.

For the **Appetite** feature, normal and poor appetite were indicated by 'Y' and 'N' respectively. A similar strategy was used for other conditions such as **Dry Mouth, Night Urination, Constipation, Acne**, and **Easily Irritable,** with the presence being noted as 'Y'. Urination Status was transformed into numerical data with ‘Y’, ‘N’ corresponding to whether they have night urination. Other binary features like **Sweating, Dizziness, Palpitations,** and **Chest Tightness** were also replaced with 'Y' for the presence of the condition.

This meticulous data processing stage was crucial in preparing the dataset for ML modeling. By transforming the original medical case data into a format that can be easily understood by ML algorithms, we have laid a solid foundation for the next stages of our research. This involves building a predictive model to recommend TCM prescriptions for individuals suffering from insomnia. The comprehensive and systematic approach to data processing ensures the integrity and richness of the original data, while facilitating efficient data analysis and ML model development.

3. **Identify and remove duplicates**:

To identify and remove duplicates, we have taken the following steps:

Identify potential duplicates: We manually reviewed the data to identify rows or observations that have the same values across all relevant attributes or variables. Compare potential duplicates: Compare the potential duplicates to ensure that they are in fact duplicates. For example, in a dataset about Chinese medicine and insomnia, the variables of interest may include the patient's age, sex, medical history, symptoms, type of Chinese medicine treatment received, duration of treatment, and outcomes. Check whether two or more observations have identical values for all these variables.

Decide which duplicate to keep: If duplicates are found, it may be necessary to decide which duplicate to keep. This can depend on the specific context and objectives of the analysis. For instance, in a study comparing different Chinese medicine treatments for insomnia, it may be preferable to keep the observation with the most complete information about the treatment received and outcomes measured.

Remove duplicates: After the decision was made, we removed the duplicates from the dataset. We did this by manually deleting the duplicate rows from the dataset.

By removing duplicates from a dataset about Chinese medicine and insomnia, the analysis can be more accurate and reliable, enabling insights to be gained with greater confidence and accuracy. It is particularly important in research studies to avoid double-counting cases, which can lead to inflated effect sizes or biased conclusions.

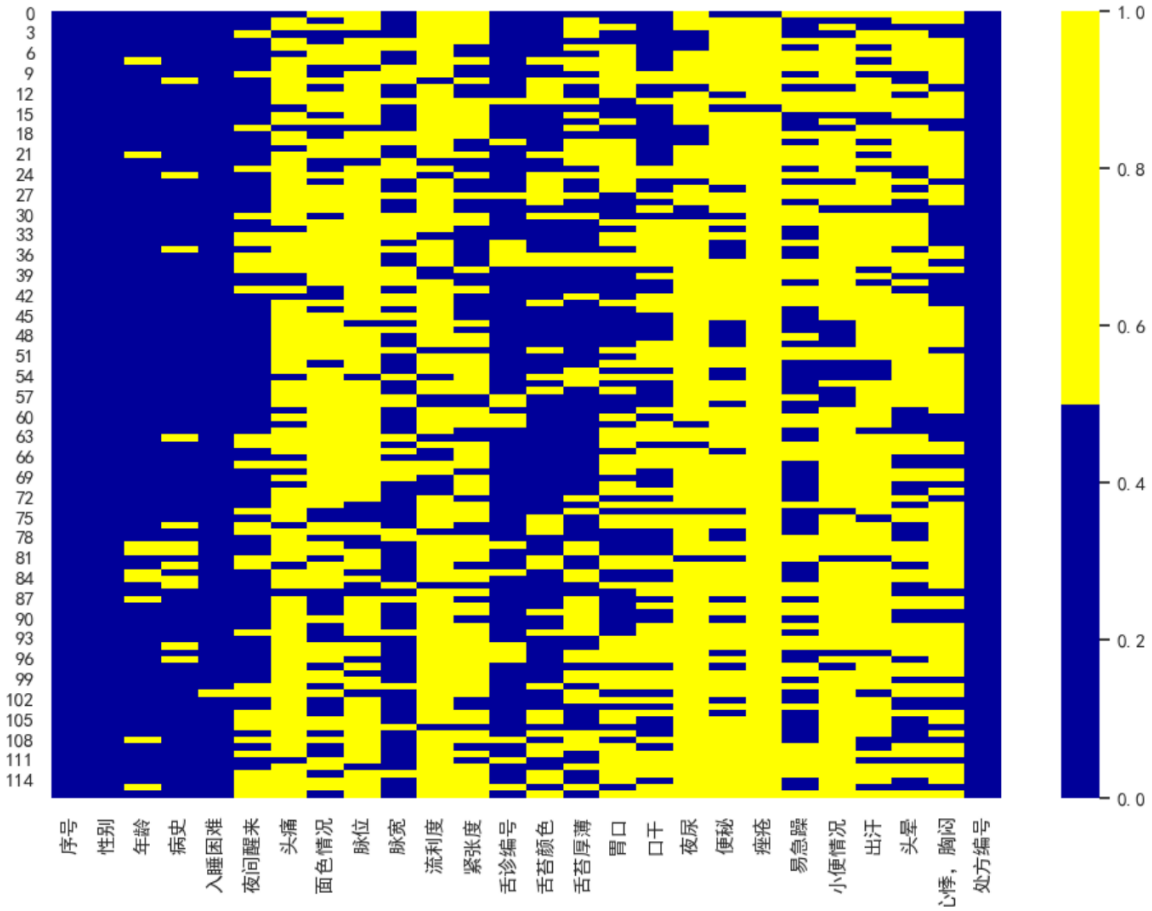
4. **Handle missing data**

We tried to find missing values in a dataset using Python and the Pandas library. Specifically, the code uses the **isnull()** function to identify all the missing values in the dataset, and then the **sum()** function is used to count the number of missing values in each column. The resulting counts are then divided by the length of the dataset and multiplied by 100 to calculate the percentage of missing values in each column. Finally, a new DataFrame is created that includes the count and percentage of missing values for each column, and this DataFrame is printed to the console.

Specific columns that are being analyzed for missing values in this dataset. These columns include Gender, Age, Medical History, Difficulty Falling Asleep, Waking Up at Night or not, Headache, Facial Color, Pulse Position, Pulse Width, Fluency, Tension, Tongue

Diagnosis, Tongue Coating Color, Tongue Coating Thickness, Appetite, Dry Mouth, Night Urination or not, Constipation or not, Acne, Easily Irritable, Urination Status, Sweating, Dizziness, Palpitations or not, and Chest Tightness. By identifying the missing values in each of these columns, researchers or analysts can better understand the quality and completeness of the data, and potentially take steps to address any missing or incomplete data.

Then we used the seaborn library to create a heatmap visualization of missing values in a Pandas DataFrame. The code first sets the font to "simhei" to support Chinese characters in the visualization. Two colors are specified for the heatmap - blue represents non-missing values, and yellow represents missing values. The **sns.heatmap()** function is called with the **data.isnull()** DataFrame as input, which identifies all missing values in the dataset. The resulting heatmap is displayed using **plt.show().**



A subsequent step in the data was processing pipeline. The code drops some columns that are deemed unnecessary for the analysis. Specifically, the data DataFrame was modified using

the **drop()** function to remove the '***Dysmenorrhea***', '***Menstrual Description***','***Facial Color***','***pulse condition***','***Tongue Diagnosis***','***Tongue coating***','***Urination***, and '***Prescription***' columns. These columns were dropped because they contained too many missing values to be useful.

Then we used the heatmap visualization step using the modified data DataFrame, which had fewer columns than the original dataset. The same color scheme and font settings are used as in the previous step. The resulting heatmap reflected the reduced number of missing values in the modified dataset. By visualizing the distribution of missing values in this way, we could gain insights into the quality and completeness of the data and identify any potential issues or biases that need to be addressed.

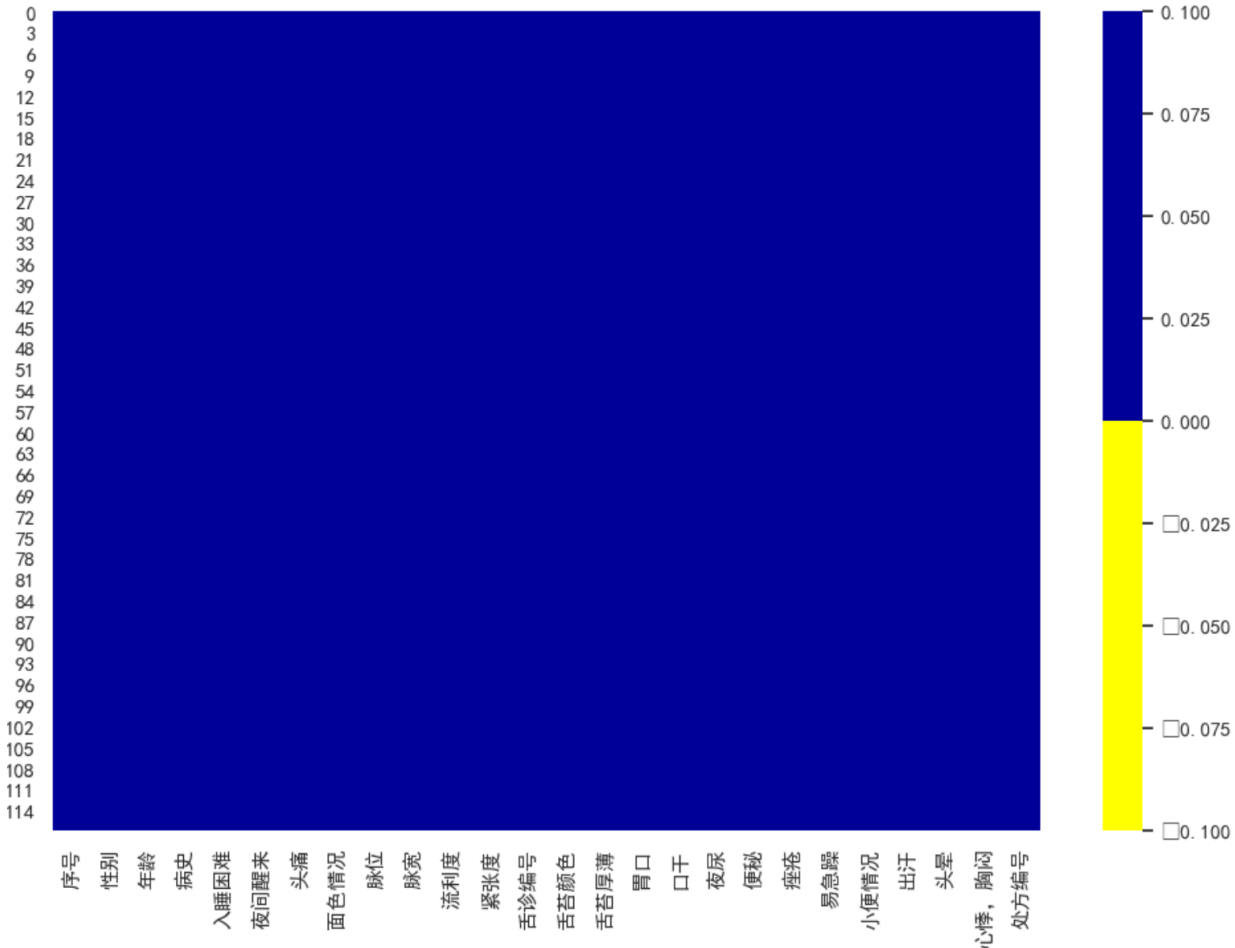


Figure 1

(3) Modeling:

1. Data Pre-processing

First we read the cleaned data again, and then we look at the data type.

```
Data columns (total 25 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Gender                          117 non-null    int64
 1   Age                             117 non-null    int64
 2   Medical Hisotry                 117 non-null    int64
 3   Difficulty Falling Asleep       117 non-null    int64
 4   Waking Up at Night              117 non-null    int64
 5   Headache                        117 non-null    int64
 6   Facial Color                    117 non-null    object
 7   Pulse Position                  117 non-null    object
 8   Pulse Width                     117 non-null    object
 9   Fluency                         117 non-null    object
 10  Tension                         117 non-null    object
 11  Tongue Diagnosis Number         117 non-null    object
 12  Tongue Coating Color            117 non-null    object
 13  Tongue Coating Thickness        117 non-null    object
 14  Appetite                        117 non-null    int64
 15  Dry Mouth                       117 non-null    int64
 16  Night Urination                 117 non-null    int64
 17  Constipation                    117 non-null    int64
 18  Acne                            117 non-null    int64
 19  Easily Irritable                117 non-null    int64
 20  Urinltion Stltus                117 non-null    int64
 21  Sweating                        117 non-null    int64
 22  Dizziness                       117 non-null    int64
 23  Palpitations, Chest Tightness   117 non-null    int64
 24  Prescription Number             117 non-null    object
dtypes: int64(16), object(9)
```

Figure 2

Since clustering analysis does not allow the existence of object types, we designed a converter to convert the columns in the original data to int.

```
RangeIndex: 117 entries, 0 to 116
Data columns (total 20 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Gender                      117 non-null    int64
 1   Age                         117 non-null    int64
 2   Medical Hisotry             117 non-null    int64
 3   Difficulty Falling Asleep   117 non-null    int64
 4   Waking Up at Night          117 non-null    int64
 5   Headache                    117 non-null    int64
 6   Facial Color                117 non-null    int32
 7   Pulse Width                 117 non-null    int32
 8   Fluency                     117 non-null    int32
 9   Tension                     117 non-null    int32
 10  Tongue Diagnosis Number     117 non-null    int32
 11  Tongue Coating Color        117 non-null    int32
 12  Tongue Coating Thickness    117 non-null    int32
 13  Appetite                    117 non-null    int64
 14  Dry Mouth                   117 non-null    int64
 15  Night Urination             117 non-null    int64
 16  Easily Irritable            117 non-null    int64
 17  Sweating                    117 non-null    int64
 18  Dizziness                   117 non-null    int64
 19  Prescription Number         117 non-null    object
dtypes: int32(7), int64(12), object(1)
```

Figure 3

2. Random Forest

We used a decision tree. This code fits a decision tree classifier using the Random Forest Classifier class from scikit-learn. It sets the criterion to "entropy", which means the decision tree will use the information gain method to decide which features to split at each node.

After fitting the decision tree, the code uses the plot_tree function from the scikit-learn tree module to visualize the first tree in the random forest. This tree is stored in clf.estimators_[0]

. The feature_names parameter is set to the column names of the input features, and class_names is set to the unique values of the target variables. The resulting graph shows a decision tree, with nodes representing splits of the input features and leaves representing predicted values of the target variable.

We also used random forests. The code works with a random forest classifier using the same RandomForestClassifier class from scikit-learn. The random_state parameter is set to 41, which means that the random forest will be initialized with a fixed seed, ensuring the same results each time the code is run.

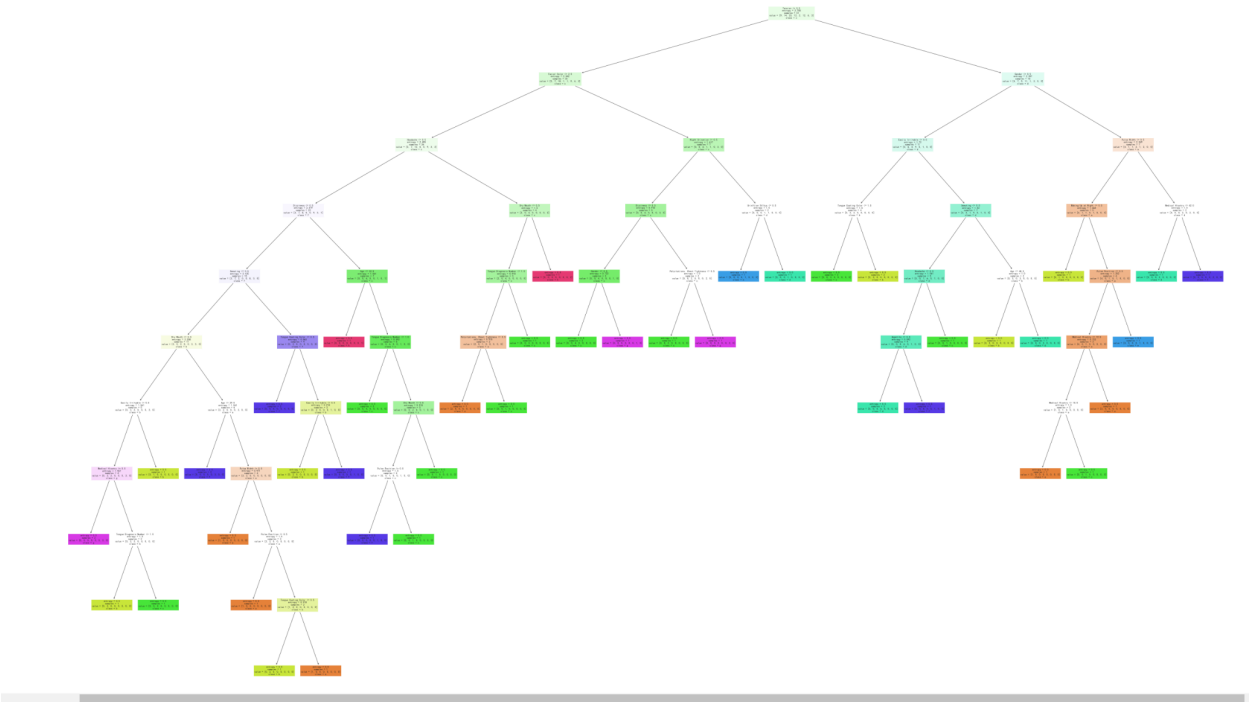


Figure 4

Understanding the essential features in a decision tree classifier is critical to developing an accurate model that can effectively predict outcomes of interest. In this case, a decision tree classifier trained on data related to insomnia found that the most important features were age, medical history, tongue diagnosis number, facial color, pulse width, and tongue coating color.

Age is a critical characteristic that can help predict an individual's likelihood of insomnia. As people age, their sleep patterns may change, making them more prone to insomnia. Therefore, age can be a strong predictor of insomnia in the model.

Medical history is another important characteristic that can help identify those at risk for insomnia. Medical conditions such as depression, anxiety, and chronic pain can significantly affect an individual's sleep patterns and increase the likelihood of insomnia. Therefore, medical history is an important feature that can help the model identify individuals at risk for insomnia.

Tongue diagnosis number, facial color, pulse width, tongue coating color and other characteristics related to traditional Chinese medicine. These characteristics can help identify individuals experiencing health problems that can interfere with sleep and lead to insomnia. In traditional Chinese medicine, the tongue reflects a person's overall health, with features such as tongue coating color, facial color, and pulse width providing valuable information about an individual's health. Therefore, these features may be useful indicators for identifying individuals at risk for insomnia.

On the other hand, acne was found to be the least important feature in the model, with an importance score of 0. This means that acne did not contribute much to predicting insomnia. While acne might be an underlying feature in other conditions, it did not significantly impact the model's ability to predict insomnia.

	Feature	Importance
1	Age	0.108813
2	Medical Hisotry	0.099654
11	Tongue Diagnosis Number	0.095410
6	Facial Color	0.078678
8	Pulse Width	0.069892
12	Tongue Coating Color	0.057012
22	Dizziness	0.049948
19	Easily Irritable	0.047586
10	Tension	0.044298
15	Dry Mouth	0.042272
0	Gender	0.040566
13	Tongue Coating Thickness	0.031445
9	Fluency	0.030854
16	Night Urination	0.028302
21	Sweating	0.026911
4	Waking Up at Night	0.026665
5	Headache	0.024436
23	Palpitations, Chest Tightness	0.022636
14	Appetite	0.021678
7	Pulse Position	0.020242
17	Constipation	0.019112
20	Urinltion Stltus	0.012356
3	Difficulty Falling Asleep	0.001236
18	Acne	0.000000

Figure 5

3. Decision Tree Model

We created a list to store the cross-validation scores for different values of the maximum depth of the decision tree classifier. We use the NumPy arrange function to create an array of integers from 1 to 14, representing different depths of the tree. We then iterated through this array, creating a decision tree classifier object with a specific depth and entropy as the split criterion. We then fitted the classifier to the training data and evaluate its performance using cross-validation with 10 folds. The mean cross-validation score for each depth is stored in the scores list. Finally, we plot the scores against the depth values to visualize the relationship between them and identify the optimal value of the maximum depth that yields the highest accuracy score. The best max depth by cross-validation is 14 and the optimal choice is 8.

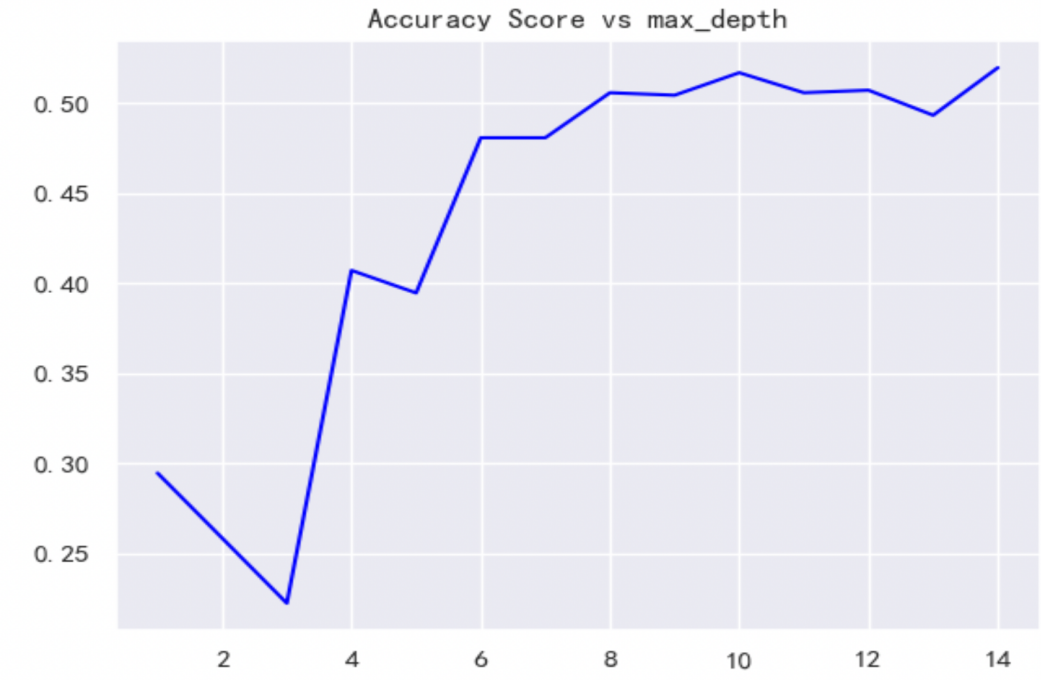


Figure 6

We used a Decision Tree Classifier object with specific parameters. The "**random_state**" parameter was used to ensure that the results of the decision tree were reproducible across multiple runs. The "**criterion**" parameter specified the quality of the split. "**Entropy**" was used here, which measured the impurity of a group of samples based on the probability of their belonging to a specific class. After the Decision Tree Classifier object was initialized, it was fitted to the training data using the "**fit**" method. This process involved recursively splitting the

data into subsets based on the values of the features until the resulting subsets contain only samples of the same class.

Apart from that, we used the plot_tree function from the "**sklearn.tree"** module to visualize the decision tree that was trained on the data. The "**fn**" variable contained the names of the features in the "**X_test"** dataset, while the "**cn**" variable contained the unique values of the target variable "**y_test"**. The plot_tree function was then called with the trained decision tree model as the first argument and the feature and class names as the second and third arguments, respectively. Then we used "filled" parameter to color the nodes in the tree based on the majority class in the subset of samples at that node. And we used the "**fontsize**" parameter to adjust the size of the text in the plot. The resulting plot showed the decision tree with each node representing a split based on a feature value, and the leaf nodes representing the predicted class for a given sample.

In the given tree model, each node represented a condition or rule based on one of the features. The topmost node, known as the root node, represented the initial split based on the condition "pulse width < 0.5". This meant that the data was divided into two branches: one for samples where the pulse width was less than 0.5 and another for samples where the pulse width were greater than or equal to 0.5.

The branches that emerge from the split represent the different possible outcomes or decisions based on the condition. In this case, the left branch represented the "yes" outcome, while the right branch represented the "no" outcome. Each subsequent node further split the data based on additional conditions, creating a hierarchical structure.

The color of the split nodes indicated the level of impurity or entropy. A darker color suggests a higher level of purity, meaning that the samples in that subset were predominantly of the same class. Conversely, a lighter color indicated a lower purity, suggesting a mix of different classes in that subset.

At the end of each branch, we reached a leaf node, which represented the final decision or prediction. In this context, the leaf node with the highest entropy represented the most probable prescription based on the given features and the training data.

By following the path from the root node to a specific leaf node, we can determine the sequence of conditions that lead to that particular prescription. This tree model provided a visual

representation of the decision-making process and helped us understand how the features contribute to the final outcome.

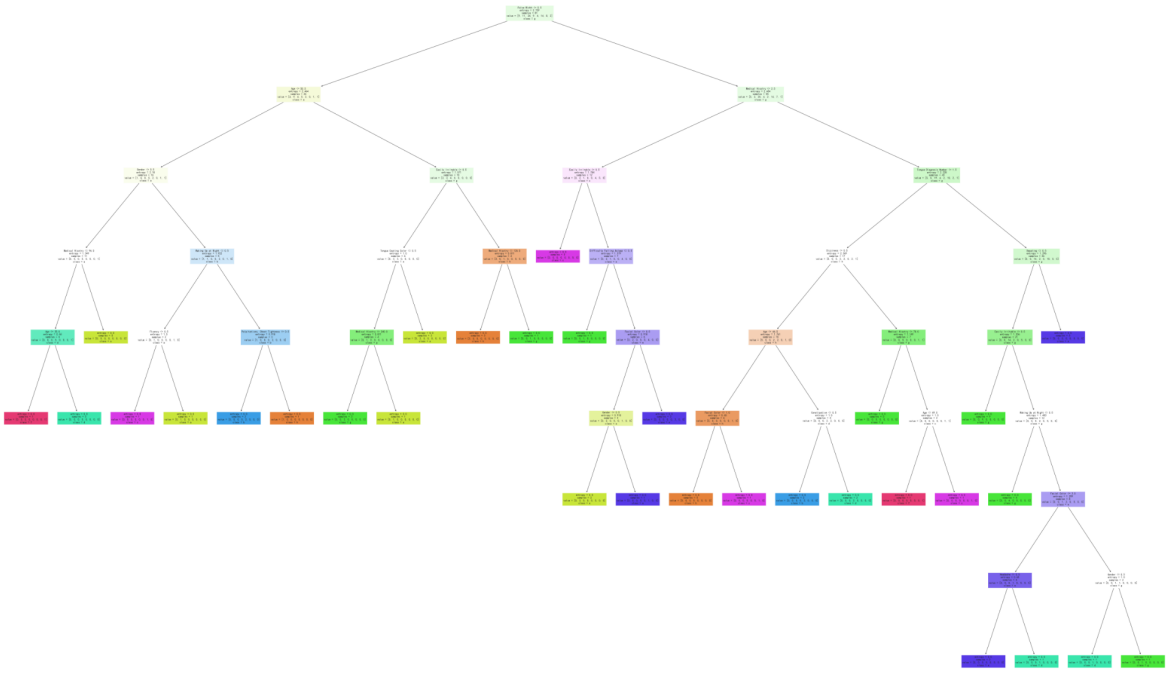


Figure 7

4. Logistic regression

For model 3 we chose logistic regression and divided the original data data into 70% training and 30% test data. The results of the analysis in logistic regression are as follows.

	precision	recall	f1-score	support
a	0.33	0.25	0.29	4
b	1.00	0.25	0.40	4
c	0.44	0.64	0.52	11
d	0.25	0.25	0.25	4
e	1.00	0.50	0.67	2
f	0.60	0.75	0.67	4
g	0.33	0.50	0.40	4
h	0.00	0.00	0.00	3
accuracy			0.44	36
macro avg	0.49	0.39	0.40	36
weighted avg	0.47	0.44	0.42	36

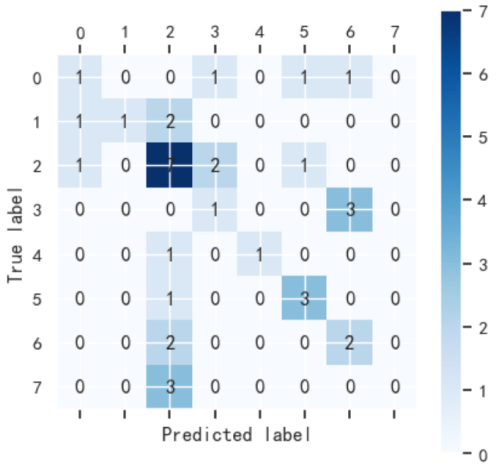


Figure 8, 9

The main diagonal line in the confusion matrix indicates the number of correct model predictions, while the other elements indicate the way the model predicts incorrectly. For example, the element in row i, column j indicates the number of instances where the true category is i but the model predicts it to be category j. Also in the first figure, we can analyze to get the accuracy and F-1 scores of each pharmacy in the test.

(4)Performance

We calculated ACC mean and ACC STD for each of the three models we used, so we choose to use Decision Tree Classifier here because among the three models Decision Tree Classifier has the highest ACC Mean and the lowest ACC STD, so Decision Tree Classifier Classifier model is the optimal clustering model in this scenario.

	Algorithm	Accuracy Mean	Accuracy STD
0	Logistic Regression	38.47	13.37
1	Decision Tree Classifier	54.44	10.41
2	Random Forest	54.31	11.19

Figure 10

(5)Business Concerns and Costs

When this model comes to an incorrect conclusion, it may result in unexpected business costs. For instance, if our model recommends prescription 1, but the patient actually requires prescription 2, it could prevent the patient's insomnia from being cured. Typically, in traditional Chinese medicine, during the initial consultation, the doctor will prescribe a course of medicine for the patient, usually for seven days. If the insomnia does not improve, the doctor will change the prescription during the second consultation. However, traditional Chinese medicine primarily uses natural plant-based ingredients, so even if the patient takes the wrong prescription, it is unlikely to result in death. Therefore, it is relatively safe. The business cost for us may be the need for an additional course of medication and the extra time required.

In addition to the potential costs of prescribing the wrong medication, there are other possible business costs associated with using machine learning models in healthcare. One significant issue is the potential for biased or inaccurate results, which can lead to misdiagnosis or inappropriate treatment. This could result in prolonged illness, more frequent visits to healthcare providers, and increased healthcare expenses.

Another important consideration is the risk of data breaches or other security concerns. As machine learning models rely on large amounts of sensitive patient data, there is always a risk of data breaches or other security vulnerabilities. This could result in patient information being compromised, leading to potential legal and reputational damage.

Appendix

Jingfangpai. https://www.jingfangpai.cn/

[1]:
```
#Standard libraries for data analysis:
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# sklearn modules for data preprocessing:
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split

#sklearn modules for Model Selection:
from sklearn import svm, tree, linear_model, neighbors
from sklearn import naive_bayes, ensemble
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

#sklearn modules for Model Evaluation & Improvement:
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn import feature_selection
from sklearn import model_selection
from sklearn import metrics
from sklearn.metrics import auc, roc_auc_score, roc_curve
import seaborn as sns
import warnings
warnings.simplefilter("ignore")
```

[2]:
```
data = pd.read_excel("ml2_dataset.xlsx")
data.head
```

```
})

# Sort the DataFrame by importance
feature_importances = feature_importances.sort_values(by='Importance', ascending=False)

print(feature_importances)
```

	Feature	Importance
1	Age	0.108813
2	Medical Hisotry	0.099654
11	Tongue Diagnosis Number	0.095410
6	Facial Color	0.078678
8	Pulse Width	0.069892
12	Tongue Coating Color	0.057012
22	Dizziness	0.049948
19	Easily Irritable	0.047586
10	Tension	0.044298
15	Dry Mouth	0.042272
0	Gender	0.040566
13	Tongue Coating Thickness	0.031445
9	Fluency	0.030854
16	Night Urination	0.028302
21	Sweating	0.026911
4	Waking Up at Night	0.026665
5	Headache	0.024436
23	Palpitations, Chest Tightness	0.022636
14	Appetite	0.021678
7	Pulse Position	0.020242
17	Constipation	0.019112
20	Urinltion Stltus	0.012356
3	Difficulty Falling Asleep	0.001236
18	Acne	0.000000

In [33]:
```
dtc = DecisionTreeClassifier(random_state = 41, criterion="entropy")
dtc_model=dtc.fit(X_train,y_train)
```

In [34]:
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
fn = X_test.columns.values
cn = y_test.unique()
plt.figure(figsize = (80,50))
plot_tree(dtc_model, feature_names = fn, class_names = cn, filled = True, fontsize=10);
```

Random Forest and Decision Tree

In [29]:
```
# Label Encode Categorical Features

#Create a label encoder object
le = LabelEncoder()
# Encode for all categorical features
le_count = 0
for col in data.columns[:-1]: # -1 because the last column in this example is the target variable
    if data[col].dtype == 'object':
        le.fit(data[col])
        data[col] = le.transform(data[col])
        le_count += 1
print('{} columns were label encoded.'.format(le_count))
```

```
8 columns were label encoded.
```

In [30]:
```
target = 'Prescription Number'
X = data.drop(columns=target)
y = data[target]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,  test_size=0.3, random_state=41)
```

In [31]:
```
# Create a decision tree classifier and fit it to the training data
clf = RandomForestClassifier(random_state=41, criterion="entropy")
clf = clf.fit(X_train, y_train)

# Visualize the decision tree
fn = X_test.columns.values
cn = y_test.unique()
plt.figure(figsize = (80,50))
_ = tree.plot_tree(clf.estimators_[0],  # We are visualizing only one tree from the forest here
                   feature_names=X.columns,
                   class_names=[str(i) for i in clf.classes_],  # class names should be the unique values in y
                   filled=True)


plt.show()  # Show the plot
```

```
Y    115
y      1
N      1
Name: 入睡困难, dtype: int64
```

n [14]:
```
data[['面色情况']]=data[['面色情况']].fillna("9.0")
```

n [15]:
```
counts = data['面色情况'].value_counts(dropna=False)
print(counts)
```

```
9.0    79
5.0    15
3.0    10
2.0     6
1.0     5
4.0     2
Name: 面色情况, dtype: int64
```

n [16]:
```
data[['舌诊编号']]=data[['舌诊编号']].fillna("1.0")
```

n [17]:
```
counts = data['舌诊编号'].value_counts(dropna=False)
print(counts)
```

```
3.0    49
2.0    22
1.0    19
1.0    16
4.0    11
Name: 舌诊编号, dtype: int64
```

n [18]:
```
data[['舌苔颜色']]=data[['舌苔颜色']].fillna("1.0")
```

n [19]:
```
counts = data['舌苔颜色'].value_counts(dropna=False)
print(counts)
```

```
1.0    47
1.0    34
2.0    30
3.0     6
Name: 舌苔颜色, dtype: int64
```

n [20]:
```
data[['舌苔厚薄']]=data[['舌苔厚薄']].fillna("1.0")
```

In [21]:
```
counts = data['舌苔颜色'].value_counts(dropna=False)
print(counts)
```

```
1.0    47
1.0    34
2.0    30
3.0     6
Name: 舌苔颜色, dtype: int64
```

In [22]:
```
data[['小便情况']]=data[['小便情况']].fillna("1.0")
```

In [23]:
```
counts = data['小便情况'].value_counts(dropna=False)
print(counts)
```

```
1.0    96
1.0    15
2.0     4
4.0     2
Name: 小便情况, dtype: int64
```

In [24]:
```
data[['脉位','脉宽','流利度','紧张度']]=data[['脉位','脉宽','流利度','紧张度']].fillna("9.0")
```

In [25]:
```
sns.set(font="simhei")
colors = ['#ffff00','#000099']  # specify the colors - yellow is missing. blue is not missing.
plt.figure(figsize=(12, 8))
sns.heatmap(data.isnull(), cmap = sns.color_palette(colors))
plt.show()
```

```
})

# Sort the DataFrame by importance
feature_importances = feature_importances.sort_values(by='Importance', ascending=False)

print(feature_importances)
```

```
                          Feature  Importance
1                             Age    0.108813
2                  Medical Hisotry    0.099654
11         Tongue Diagnosis Number    0.095410
6                    Facial Color    0.078678
8                     Pulse Width    0.069892
12           Tongue Coating Color    0.057012
22                      Dizziness    0.049948
19               Easily Irritable    0.047586
10                        Tension    0.044298
15                     Dry Mouth    0.042272
0                         Gender    0.040566
13       Tongue Coating Thickness    0.031445
9                         Fluency    0.030854
16                Night Urination    0.028302
21                      Sweating    0.026911
4               Waking Up at Night    0.026665
5                        Headache    0.024436
23  Palpitations, Chest Tightness    0.022636
14                       Appetite    0.021678
7                  Pulse Position    0.020242
17                   Constipation    0.019112
20              Urinltion Stltus    0.012356
3       Difficulty Falling Asleep    0.001236
18                           Acne    0.000000
```

In [33]:
```
dtc = DecisionTreeClassifier(random_state = 41, criterion="entropy")
dtc_model=dtc.fit(X_train,y_train)
```

In [34]:
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
fn = X_test.columns.values
cn = y_test.unique()
plt.figure(figsize = (80,50))
plot_tree(dtc_model, feature_names = fn, class_names = cn, filled = True, fontsize=10);
```

In [51]:
```
# Initialize classifiers in consideration
models = []
models.append(('Logistic Regression', LogisticRegression()))
models.append(('Decision Tree Classifier', DecisionTreeClassifier(criterion = 'entropy')))
models.append(('Random Forest', RandomForestClassifier(n_estimators=100, criterion = 'entropy')))

#Evaluating Model Results:
acc_results = []
auc_results = []
names = []
# set table to table to populate with performance results
col = ['Algorithm', 'Accuracy Mean', 'Accuracy STD']
model_results = pd.DataFrame(columns=col)

# Evaluate each model using k-fold cross-validation:
i = 0
for name, model in models:
    kfold = model_selection.KFold(n_splits=10)
    # accuracy scoring:
    cv_acc_results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold, scoring='accuracy')
    # roc_auc scoring:
    cv_auc_results = model_selection.cross_val_score(model, X_train, y_train, cv=kfold, scoring='roc_auc')
    acc_results.append(cv_acc_results)
    auc_results.append(cv_auc_results)
    names.append(name)
    model_results.loc[i] = [name,
                            round(cv_acc_results.mean()*100, 2),
                            round(cv_acc_results.std()*100, 2)
                            ]

    i += 1

model_results
```

Out[51]:

	Algorithm	Accuracy Mean	Accuracy STD
0	Logistic Regression	38.47	13.37
1	Decision Tree Classifier	54.44	10.41
2	Random Forest	54.31	11.19

In [42]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

lr = LogisticRegression()

lr.fit(X_train, y_train)

y_pred = lr.predict(X_test)

print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)

plt.matshow(cm, cmap=plt.cm.Blues)
plt.colorbar()

for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        plt.text(x=j, y=i, s=cm[i, j], va='center', ha='center')

plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.show()
```