# Лабораторная работа №2

## по дисциплине *Моделирование инженерных задач*

Работу выполнил: студент гр. **М1О-414Бки-19** *Дун Бинь*

**Вариант 4**

Задачей лабораторной работы является построение моделей для классификации белых или красных вин

## Красное вино

In [ ]:
```
# Повторение первых шагов из 1 лабораторной работы
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd

dataset = pd.read_csv('datasets/winequality-red.csv', sep=';')

quality_bins = (2,6.5,8)    #Больше 6.5 - отличное вино, иначе - обычное вино
qualities = ['normal','elite']
categories = pd.cut(dataset['quality'], quality_bins, labels = qualities)
dataset['quality'] = categories
```

In [ ]:
```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler, OrdinalEncoder

preprocessing = ColumnTransformer(
    [('encoder', OrdinalEncoder(), ['quality'])],
    remainder=(MinMaxScaler()),    #Остальные столбцы масштабируем
    verbose_feature_names_out=False)    #Не добавляем префикс к названиям столби
preprocessing.set_output(transform = 'pandas')

dataset = preprocessing.fit_transform(dataset)
dataset.head()
```

Out[ ]:

| | quality | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | p |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1.0 | 0.247788 | 0.397260 | 0.00 | 0.068493 | 0.106845 | 0.140845 | 0.098940 | 0.567548 | 0.6062 |
| **1** | 1.0 | 0.283186 | 0.520548 | 0.00 | 0.116438 | 0.143573 | 0.338028 | 0.215548 | 0.494126 | 0.3622 |
| **2** | 1.0 | 0.283186 | 0.438356 | 0.04 | 0.095890 | 0.133556 | 0.197183 | 0.169611 | 0.508811 | 0.4094 |
| **3** | 1.0 | 0.584071 | 0.109589 | 0.56 | 0.068493 | 0.105175 | 0.225352 | 0.190813 | 0.582232 | 0.3307 |
| **4** | 1.0 | 0.247788 | 0.397260 | 0.00 | 0.068493 | 0.106845 | 0.140845 | 0.098940 | 0.567548 | 0.6062 |

После обработки данные находятся в промежутке от 0 до 1, что упрощает работу с ними. Помимо этого, вместо строкого представления качество теперь представлено как числа 0 и 1, что также упрощает работу.

Перед тем, как подавать данные на вход моделям для обучения, данные делятся на выборки:

In [ ]:
```
from sklearn.model_selection import StratifiedShuffleSplit

labels = dataset['quality']
data = dataset.copy().drop('quality', axis=1)

stratif_split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=4
for train, test in stratif_split.split(data, labels):
    pass

X_test = data.iloc[test]
y_test = labels.iloc[test]

X_train = data.iloc[train]
y_train = labels.iloc[train]
```

Чтобы не повторять все действия для каждой модели, действия объединим в одну функцию:

In [ ]:
```
import pickle
from scipy.stats import randint, uniform
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, prec
from sklearn.model_selection import GridSearchCV
import warnings
from sklearn.exceptions import ConvergenceWarning
warnings.filterwarnings(action='ignore', category=ConvergenceWarning)

def learn_save_score(clf, params, X_train, y_train, X_test, y_test, path, name):
    result_score = pd.DataFrame({'scores' : ['Accuracy', 'Recall', 'Precision',

    model = GridSearchCV(clf, params)
    model.fit(X_train, y_train)

    with open(f"saved_models/{path}/{name}.pkl", "wb") as file:
        pickle.dump(model, file, protocol=3)
    with open(f"saved_models/{path}/{name}.txt", "w") as file:
        file.write(str(model.best_params_))

    y_pred = model.predict(X_test)  # Результаты показанные моделью
    y_true = np.array(y_test)   # Истинные результаты из тестовой выборки

    result_score[name] = [  accuracy_score(y_true, y_pred), recall_score(y_true,
                            precision_score(y_true, y_pred), roc_auc_score(y_tru

    print(f'Confusion matrix:\n {confusion_matrix(y_true, y_pred)}\n')
    return result_score
```

### Логистическая регрессия

In [ ]:
```
from sklearn.linear_model import LogisticRegression
log_regression = LogisticRegression(max_iter=500)

params = [
    {'C': [1, 25], 'solver': ['lbfgs'], 'penalty': ['l2']},
    {'C': [1, 25], 'solver': ['liblinear'], 'penalty': ('l1', 'l2')},
    {'C': [1, 25], 'solver': ['saga'], 'penalty': ['elasticnet'], 'l1_ratio': [0
]

learn_save_score(log_regression, params, X_train, y_train, X_test, y_test, 'red'
```

```
Confusion matrix:
 [[ 16  27]
 [  9 268]]
```

Out[ ]:

| | scores | logreg |
|---|---|---|
| **0** | Accuracy | 0.887500 |
| **1** | Recall | 0.967509 |
| **2** | Precision | 0.908475 |
| **3** | ROC AUC curve | 0.669801 |

## Метод опорных векторов

In [ ]:
```
from sklearn.svm import SVC

svm = SVC(max_iter=500)

params = [
    {'C': [1, 25], 'kernel': ['poly', 'rbf', 'sigmoid'], 'gamma': ['scale', 'aut
]

learn_save_score(svm, params, X_train, y_train, X_test, y_test, 'red', 'svm')
```

```
Confusion matrix:
 [[ 19  24]
 [ 11 266]]
```

Out[ ]:

| | scores | svm |
|---|---|---|
| **0** | Accuracy | 0.890625 |
| **1** | Recall | 0.960289 |
| **2** | Precision | 0.917241 |
| **3** | ROC AUC curve | 0.701075 |

## К ближайших соседей

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()

params = [
    {'n_neighbors': [2, 10], 'weights': ['uniform', 'distance'], 'algorithm': ['
```

```
]

learn_save_score(knn, params, X_train, y_train, X_test, y_test, 'red', 'knn')
```

```
Confusion matrix:
 [[ 24  19]
 [ 13 264]]
```

Out[ ]:

| | scores | knn |
|---|---|---|
| **0** | Accuracy | 0.900000 |
| **1** | Recall | 0.953069 |
| **2** | Precision | 0.932862 |
| **3** | ROC AUC curve | 0.755604 |

## Наивный Байесовский классификатор

In [ ]:

```
from sklearn.naive_bayes import GaussianNB

bayes = GaussianNB()

params = [
    {'var_smoothing': [1e-12, 1e-2]}
]

learn_save_score(bayes, params, X_train, y_train, X_test, y_test, 'red', 'bayes'
```

```
Confusion matrix:
 [[ 25  18]
 [ 40 237]]
```

Out[ ]:

| | scores | bayes |
|---|---|---|
| **0** | Accuracy | 0.818750 |
| **1** | Recall | 0.855596 |
| **2** | Precision | 0.929412 |
| **3** | ROC AUC curve | 0.718496 |

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier()

params = [
    {'n_estimators': [100, 400], 'criterion': ['gini', 'entropy', 'log_loss'], '
]

learn_save_score(forest, params, X_train, y_train, X_test, y_test, 'red', 'fores
```

```
Confusion matrix:
 [[ 24  19]
 [  7 270]]
```

Out[ ]:

| | scores | forest |
|---|---|---|
| **0** | Accuracy | 0.918750 |
| **1** | Recall | 0.974729 |
| **2** | Precision | 0.934256 |
| **3** | ROC AUC curve | 0.766434 |

В итоге для обоих датасетов лучший результат показало применение случайного леса. С другой стороны, именно его обучение занимает больше всего времени. Для белого вина случайный лес показал лучшие результаты по всем параметрам, однако для красного метод SVM показал лучший Recall - что показывает, что модель в данной ситуации с большей вероятностью выберет правильно хорошие вина, но при этом в эту выборку могут попасть и плохие.

## Белое вино

In [ ]:
```
dataset = pd.read_csv('datasets/winequality-white.csv', sep=';')
quality_bins = (2,6.5,8)    #Больше 6.5 - отличное вино, иначе - обычное вино
qualities = ['normal','elite']
categories = pd.cut(dataset['quality'], quality_bins, labels = qualities)
dataset['quality'] = categories
dataset = preprocessing.fit_transform(dataset)
dataset = dataset.dropna(axis='rows')
labels = dataset['quality']
data = dataset.copy().drop('quality', axis=1)

stratif_split = StratifiedShuffleSplit(n_splits=1, test_size=0.2, random_state=4
for train, test in stratif_split.split(data, labels):
    pass

X_test = data.iloc[test]
y_test = labels.iloc[test]
X_train = data.iloc[train]
y_train = labels.iloc[train]
```

### Логистическая регрессия

In [ ]:
```
from sklearn.linear_model import LogisticRegression
log_regression = LogisticRegression(max_iter=500)

params = [
    {'C': [1, 25], 'solver': ['lbfgs'], 'penalty': ['l2']},
    {'C': [1, 25], 'solver': ['liblinear'], 'penalty': ('l1', 'l2')},
    {'C': [1, 25], 'solver': ['saga'], 'penalty': ['elasticnet'], 'l1_ratio': [0
]

learn_save_score(log_regression, params, X_train, y_train, X_test, y_test, 'whit
```

```
Confusion matrix:
 [[ 52 159]
 [ 33 735]]
```

Out[ ]:

| | scores | logreg |
|---|---|---|
| **0** | Accuracy | 0.803882 |
| **1** | Recall | 0.957031 |
| **2** | Precision | 0.822148 |
| **3** | ROC AUC curve | 0.601738 |

## Метод опорных векторов

In [ ]:
```
from sklearn.svm import SVC

svm = SVC(max_iter=500)

params = [
    {'C': [1, 25], 'kernel': ['poly', 'rbf', 'sigmoid'], 'gamma': ['scale', 'aut
]

learn_save_score(svm, params, X_train, y_train, X_test, y_test, 'white', 'svm')
```

```
Confusion matrix:
 [[ 23 188]
 [ 40 728]]
```

Out[ ]:

| | scores | svm |
|---|---|---|
| **0** | Accuracy | 0.767109 |
| **1** | Recall | 0.947917 |
| **2** | Precision | 0.794760 |
| **3** | ROC AUC curve | 0.528461 |

## К ближайших соседей

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()

params = [
    {'n_neighbors': [2, 10], 'weights': ['uniform', 'distance'], 'algorithm': ['
]

learn_save_score(knn, params, X_train, y_train, X_test, y_test, 'white', 'knn')
```

```
Confusion matrix:
 [[135  76]
 [ 46 722]]
```

Out[ ]:

| | scores | knn |
|---|---|---|
| 0 | Accuracy | 0.875383 |
| 1 | Recall | 0.940104 |
| 2 | Precision | 0.904762 |
| 3 | ROC AUC curve | 0.789957 |

## Наивный Байесовский классификатор

In [ ]:
```
from sklearn.naive_bayes import GaussianNB

bayes = GaussianNB()

params = [
    {'var_smoothing': [1e-12, 1e-2]}
]

learn_save_score(bayes, params, X_train, y_train, X_test, y_test, 'white', 'baye
```

```
Confusion matrix:
 [[150  61]
 [199 569]]
```

Out[ ]:

| | scores | bayes |
|---|---|---|
| 0 | Accuracy | 0.734423 |
| 1 | Recall | 0.740885 |
| 2 | Precision | 0.903175 |
| 3 | ROC AUC curve | 0.725893 |

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier()

params = [
    {'n_estimators': [100, 400], 'criterion': ['gini', 'entropy', 'log_loss'], '
]

learn_save_score(forest, params, X_train, y_train, X_test, y_test, 'white', 'for
```

```
Confusion matrix:
 [[129  82]
 [ 30 738]]
```

Out[ ]:

| | scores | forest |
|---|---|---|
| 0 | Accuracy | 0.885598 |
| 1 | Recall | 0.960938 |
| 2 | Precision | 0.900000 |
| 3 | ROC AUC curve | 0.786156 |

Результаты получены следующие: для обоих датасетов лучше использовать модели на основе случайных лесов. Это позволило получить точности в 91,9% и 88,6% для красного и белого вина. Такие результаты обоснованы размерами датасета - датасет с качеством красного вина больше, чем датасет с качеством белого вина, что позволило создать большие, чем у белого вина, обучающие и тестовые выборки. В итоге построенные результаты можно считать удовлетворительными, и при условии больших датасетов можно будет улучшить точность определения вина еще выше.