Binary classification

of

Poisonous Mushrooms

ZHAO SHENGXUN
L3 IA
University of Nice Sophia-Antipolis

Content

Problematic:

This is a machine learning project on the website Kaggle.

The mission of this project is to create a binary classification model to predict whether a mushroom is edible or poisonous based on its physical characteristics.

I. Data exploration

Training data: train.csv, 160Mb

1. Size: (3116945, 22)

```
print(train_set.shape)
✓ [2] < 10 毫秒
(3116945, 22)
```

2. Columns and type of data:

```
print(train_set.info())     train_set
✓ [3] 15毫秒
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3116945 entries, 0 to 3116944
Data columns (total 22 columns):
 #   Column                Dtype
---  ------                -----
 0   id                    int64
 1   class                 object
 2   cap-diameter          float64
 3   cap-shape             object
 4   cap-surface           object
 5   cap-color             object
 6   does-bruise-or-bleed  object
 7   gill-attachment       object
 8   gill-spacing          object
 9   gill-color            object
 10  stem-height           float64
 11  stem-width            float64
 12  stem-root             object
 13  stem-surface          object
 14  stem-color            object
 15  veil-type             object
 16  veil-color            object
 17  has-ring              object
 18  ring-type             object
 19  spore-print-color     object
 20  habitat               object
 21  season                object
dtypes: float64(3), int64(1), object(18)
memory usage: 523.2+ MB
None
```

3. Data preview:

```
print(train_set.head())    train_set
✓ [4] 13毫秒
   id class  cap-diameter cap-shape cap-surface cap-color  \
0   0     e          8.80         f           s         u
1   1     p          4.51         x           h         o
2   2     e          6.94         f           s         b
3   3     e          3.88         f           y         g
4   4     e          5.85         x           l         w

  does-bruise-or-bleed gill-attachment gill-spacing gill-color  ...  \
0                    f               a            c          w  ...
1                    f               a            c          n  ...
2                    f               x            c          w  ...
3                    f               s          NaN          g  ...
4                    f               d          NaN          w  ...

   stem-root  stem-surface stem-color veil-type veil-color has-ring ring-type  \
0        NaN           NaN          w       NaN        NaN        f         f
1        NaN             y          o       NaN        NaN        t         z
2        NaN             s          n       NaN        NaN        f         f
3        NaN           NaN          w       NaN        NaN        f         f
4        NaN           NaN          w       NaN        NaN        f         f

  spore-print-color habitat season
0               NaN       d      a
1               NaN       d      w
2               NaN       l      w
3               NaN       d      u
4               NaN       g      a

[5 rows x 22 columns]
```

Test data: test.csv, 103Mb

1. Size: (2077964,21)

```
print(test_set.shape)
✓ [5] < 10 毫秒
(2077964, 21)
```

2. Columns and type of data:

```
print(test_set.info())
✓ [6] < 10 毫秒
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2077964 entries, 0 to 2077963
Data columns (total 21 columns):
 #   Column                Dtype
---  ------                -----
 0   id                    int64
 1   cap-diameter          float64
 2   cap-shape             object
 3   cap-surface           object
 4   cap-color             object
 5   does-bruise-or-bleed  object
 6   gill-attachment       object
 7   gill-spacing          object
 8   gill-color            object
 9   stem-height           float64
 10  stem-width            float64
 11  stem-root             object
 12  stem-surface          object
 13  stem-color            object
 14  veil-type             object
 15  veil-color            object
 16  has-ring              object
 17  ring-type             object
 18  spore-print-color     object
 19  habitat               object
 20  season                object
dtypes: float64(3), int64(1), object(17)
memory usage: 332.9+ MB
None
```

3. Data preview:

```
print(test_set.head())
✓ [7] 11毫秒
```

```
        id  cap-diameter cap-shape cap-surface cap-color does-bruise-or-bleed  \
0  3116945          8.64         x         NaN         n                    t
1  3116946          6.90         o           t         o                    f
2  3116947          2.00         b           g         n                    f
3  3116948          3.47         x           t         n                    f
4  3116949          6.17         x           h         y                    f

  gill-attachment gill-spacing gill-color  stem-height  ... stem-root  \
0             NaN          NaN          w        11.13  ...         b
1             NaN            c          y         1.27  ...       NaN
2             NaN            c          n         6.18  ...       NaN
3               s            c          n         4.98  ...       NaN
4               p          NaN          y         6.73  ...       NaN

  stem-surface stem-color veil-type veil-color has-ring ring-type  \
0          NaN          w         u          w        t         g
1          NaN          n       NaN        NaN        f         f
2          NaN          n       NaN        NaN        f         f
3          NaN          w       NaN          n        t         z
4          NaN          y       NaN          y        t       NaN

  spore-print-color habitat season
0               NaN       d      a
1               NaN       d      a
2               NaN       d      s
3               NaN       d      u
4               NaN       d      u

[5 rows x 21 columns]
```

Data distribution

1. Null value rate on train set:

Distribution of the target variables:

```
print(train_set.isnull().mean())A
✓ [8] 1秒 375毫秒
id                      0.000000
class                   0.000000
cap-diameter            0.000001
cap-shape               0.000013
cap-surface             0.215282
cap-color               0.000004
does-bruise-or-bleed    0.000003
gill-attachment         0.168093
gill-spacing            0.403740
gill-color              0.000018
stem-height             0.000000
stem-width              0.000000
stem-root               0.884527
stem-surface            0.635514
stem-color              0.000012
veil-type               0.948843
veil-color              0.879370
has-ring                0.000008
ring-type               0.041348
spore-print-color       0.914255
habitat                 0.000014
season                  0.000000
dtype: float64
```

2. Label's distribution

```
print(train_set['class'].value_counts(normalize=True))
✓ [9] 116毫秒

class
p    0.547137
e    0.452863
Name: proportion, dtype: float64
```

No information rate: 0.547137

II. Data preprocessing

Regarding the columns with null values, those who contain 60% or more of the null values will be removed from the train set and the other null values will be replaced by the mode value (if the column is categorical) or median (if the column is numerical).

Code display (location: preprocess.py):

```
@staticmethod  1 个用法  phianke +1
def preprocess(df:pd.DataFrame)->pd.DataFrame:
    """
    Preprocess the df_train:
    1. Remove columns with more than 60% missing values.
    2. For columns with lesser missing values, replace NaN:
        - With mode for categorical columns.
        - With median for numerical columns.
    """
    threshold = 0.6

    high_missing_cols = df.columns[df.isnull().mean() > threshold]
    processed = df.drop(columns=high_missing_cols)

    for col in processed.columns:
        if processed[col].isnull().any():
            if processed[col].dtype == 'object':
                processed[col] = processed[col].fillna(processed[col].mode()[0])
            else:
                processed[col] = processed[col].fillna(processed[col].median())

    return processed
```

After that, we will take a sample of 500,000 columns from the train set and encode them with one-hot (if the column is categorical) or min-max (if the column is numerical). *Sklearn.preprocessing module is used in this process, for more extra module (other than numpy, pandas and pytorch), see Appendix.*

Code display

(position: 02_preprocessing.ipynb)

```
preprocess = Preprocess(train_set)
#take out 500,000 samples randomly to do training
sample = preprocess.processed.sample(n=50000, random_state= 0)
dataset = ProcessedDataSet(sample)
dataset()
print(dataset.sample.head(5))
print(dataset.sample.shape)
print(dataset[0])
print(dataset.X.shape)
```

(location: dataset.py)

```
def encode(self, sample):  1 个用法  phianke
    """
    encode categorical columns with one-hot encoding.
    transform numerical columns with min-max scaling.
    :param sample:
    :return:
    """
    label_array = []
    if 'class' in sample.columns:
        category = sample['class']
        features = sample.drop(["id", "class"], axis=1)

        enc_label = preprocessing.LabelEncoder()
        label_array = enc_label.fit_transform(category)
    else:
        features = sample.drop(["id"], axis=1)

    numerical_columns = features.select_dtypes(include=["int64", "float64"])
    categorical_columns = features.select_dtypes(exclude=["int64", "float64"])

    if self.feature_order is None:
        self.enc_onehot = preprocessing.OneHotEncoder(handle_unknown='ignore')
        self.enc_minmax = preprocessing.MinMaxScaler()

        onehot_array = self.enc_onehot.fit_transform(categorical_columns).toarray()
        minmax_array = self.enc_minmax.fit_transform(numerical_columns)

        self.feature_order = list(categorical_columns.columns) + list(numerical_columns.columns)
    else:
        categorical_columns = categorical_columns.reindex(
            columns=self.feature_order[:len(categorical_columns.columns)], fill_value=0)
        numerical_columns = numerical_columns.reindex(columns=self.feature_order[len(categorical_columns.columns):],
                                                      fill_value=0)

        onehot_array = self.enc_onehot.transform(categorical_columns).toarray()
        minmax_array = self.enc_minmax.transform(numerical_columns)

    feature_array = np.concatenate( arrays: [onehot_array, minmax_array], axis=1)

    return (
        torch.tensor(feature_array, dtype=torch.float32),
        torch.tensor(label_array, dtype=torch.float32)
    )
```

Please note that the variable feature_order is used to align the columns of the training set and test set in case that they do not match each other after encoding. Columns will be reindexed and filled with 0s or removed due to the situation.

The output data are converted to torch.tensor for further use.

The processed data and the encoders will be stocked in a pickle object to be retrieved anytime without being reprocessed once more.

Code display (position: 02_preprocessing.ipynb):

```python
import pickle
os.makedirs("../data", exist_ok=True)
with open('../data/processed/preprocessed_trainSet.pkl', 'wb') as f:
    pickle.dump({
        'data_set': dataset,
        'y': dataset.y,
        'X': dataset.X,
        'feature_order': dataset.feature_order,
        'enc_onehot': dataset.enc_onehot,
        'enc_minmax': dataset.enc_minmax,
    }, f)
[3]
```

III. Binary classification model

Code display (location: binaryClassificationModel.py)

```
import torch
import torch.nn as nn

class BinaryClassificationModel(nn.Module):  8 用法  ZS912719
    def __init__(self, input_dim):  ZS912719
        super(BinaryClassificationModel, self).__init__()
        self.fc1 = nn.Linear(input_dim,  out_features: 64)
        self.fc2 = nn.Linear( in_features: 64,  out_features: 32)
        self.fc3 = nn.Linear( in_features: 32,  out_features: 1)

    def forward(self, x):  1 个用法  ZS912719
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def predict(self, x):  ZS912719
        """
        Make predictions for binary classification.

        Args:
            x: Input features (torch.Tensor).

        Returns:
            Binary predictions (torch.Tensor): A tensor of 0s and 1s.
        """
        logits = self.forward(x)
        probs = torch.sigmoid(logits)
        predictions = (probs > 0.5).float()
        return predictions

    @staticmethod  2 用法  ZS912719
    def loss(x, y):
        criterion = nn.BCEWithLogitsLoss()
        return criterion(x, y)
```

This is a MLP who contains three layers totally connected with ReLU as the activation function. The predictions are made by sigmoid function.

Linear formula:

y=w.T*x+b

y^=σ(y)=1/(1+e^(-y))

prediction = 1 when probs>0.5

prediction = 0 when probs<=0.5

Forward step :

h1=ReLU(W1x+b1)

h2=ReLU(W2h1+b2)
y^=W3h2+b3
Loss function: Binary Cross-Entropy Loss

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)\right]$$

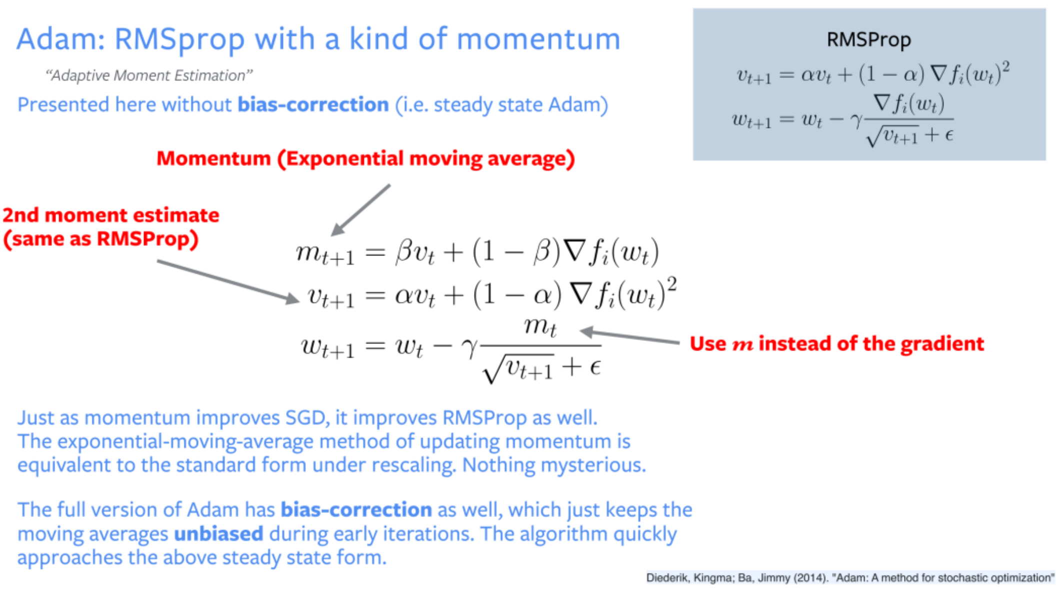
Train model (learning rate is 1e-4)**:**

```
from src.tools import train_one_epoch

model = BinaryClassificationModel(X_train.shape[1])
model = model.to(device)

dataset = list(zip(X_train, y_train))
dataloader = DataLoader(dataset, batch_size=10, shuffle=True)

metrics = {
    "loss": [],
    "accuracy": [],
}

for epoch in range(30):
    avg_loss, accuracy = train_one_epoch(model, dataloader, device)

    metrics["loss"].append(avg_loss)
    metrics["accuracy"].append(accuracy)

    print(f"Epoch {epoch+1}, Loss: {avg_loss:.4f}, Accuracy: {accuracy:.4f}")

    if (epoch + 1) % 5 == 0 or epoch == 29:
        fig, axes = plt.subplots(1, 2, figsize=(10, 4))
        axes[0].plot(range(1, len(metrics["loss"]) + 1), metrics["loss"], label="Loss")
        axes[1].plot(range(1, len(metrics["accuracy"]) + 1), metrics["accuracy"],
         label="Accuracy")

        axes[0].set_title("Loss Curve")
        axes[1].set_title("Accuracy Curve")

        axes[0].set_xlabel("Epoch")
        axes[1].set_xlabel("Epoch")

        axes[0].legend()
        axes[1].legend()

        plt.tight_layout()
        plt.show()
```

Train_one_epoch in tools.py:

```
def train_one_epoch(model, dataloader, device):  2用法  新 *
    model.train()
    total_loss = 0
    correct = 0
    total = 0
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

    for X_batch, y_batch in dataloader:
        y_batch = y_batch.unsqueeze(1).to(device)
        X_batch = X_batch.to(device)

        optimizer.zero_grad()
        logits = model(X_batch)
        loss = model.loss(logits, y_batch)

        loss.backward()
        optimizer.step()

        total_loss += loss.item()
        predictions = (torch.sigmoid(logits) > 0.5).float()
        correct += (predictions == y_batch).sum().item()
        total += y_batch.size(0)

    avg_loss = total_loss / len(dataloader)
    accuracy = correct / total
    return avg_loss, accuracy
```

Please note that the Adam algorithm is used to optimize the gradient descent process.

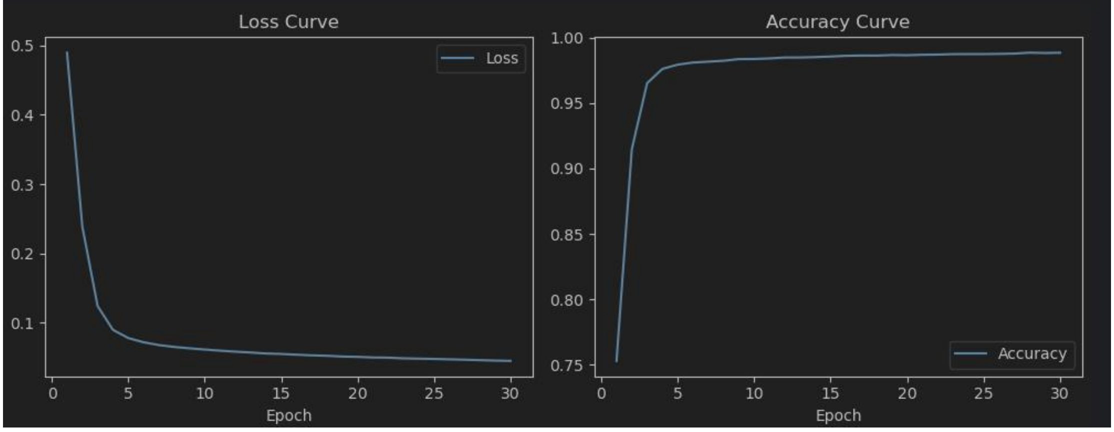


Nividia CUDA is applied in the training process.

IV. Model evaluation

As shown above in train_one_epoch.py, we can have loss curve and accuracy curve of the training set:

```
Epoch 26, Loss: 0.0474, Accuracy: 0.9877
Epoch 27, Loss: 0.0469, Accuracy: 0.9879
Epoch 28, Loss: 0.0462, Accuracy: 0.9887
Epoch 29, Loss: 0.0456, Accuracy: 0.9884
Epoch 30, Loss: 0.0452, Accuracy: 0.9886
```



And on the test set:

```
from src.tools import evaluate_model
from torch.utils.data import DataLoader
import torch
from src.binaryClassificationModel import BinaryClassificationModel

model_path ="../models/BinaryClassificationModel.pth"
model = BinaryClassificationModel(X_test.shape[1])
model.load_state_dict(torch.load(model_path, weights_only=True))

eval_data = list(zip(X_test, y_test))
eval_loader = DataLoader(eval_data, batch_size=32, shuffle=False)

metrics = evaluate_model(model, eval_loader)
print("Evaluation Metrics:", metrics)
✓ [2] 471毫秒
 Evaluation Metrics: {'accuracy': 0.9832, 'precision': 0.9848788486063035, 'recall': 0.9845201238390093, 'f1_score': 0.9846994535519126}
```

Evaluation Metrics: {
'accuracy': 0.9832,
'precision': 0.9848788486063035,
'recall': 0.9845201238390093,
'f1_score': 0.9846994535519126
}

evaluate_model in tools.py :

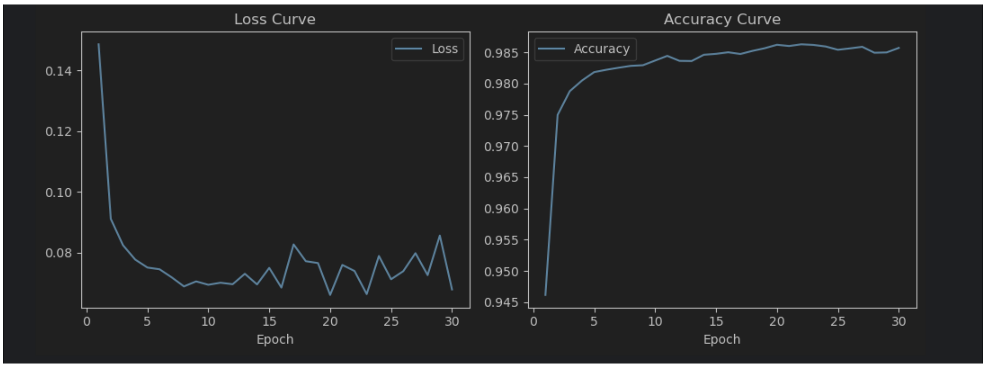
```python
def evaluate_model(model, dataloader):  2用法  新 *
    """
    Evaluate the performance of a pre-trained Binary Classification Model.

    Args:
        model (BinaryClassificationModel): The PyTorch model to evaluate.
        dataloader (DataLoader): DataLoader for the evaluation dataset.

    Returns:
        dict: A dictionary containing evaluation metrics.
    """
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for features, labels in dataloader:
            outputs = model(features)
            preds = torch.sigmoid(outputs).round()
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    accuracy = accuracy_score(all_labels, all_preds)
    precision = precision_score(all_labels, all_preds)
    recall = recall_score(all_labels, all_preds)
    f1 = f1_score(all_labels, all_preds)

    return {
        "accuracy": accuracy,
        "precision": precision,
        "recall": recall,
        "f1_score": f1
    }
```

Observation:

The loss rate declines, and the accuracy increases every epoch.

When the learning rate is greater than 1e-3, the gradient descent process will fail as the descent step is too large.

Hyper-parameter Tuning (learning rate 1e-2):



V. Prediction and Kaggle submission

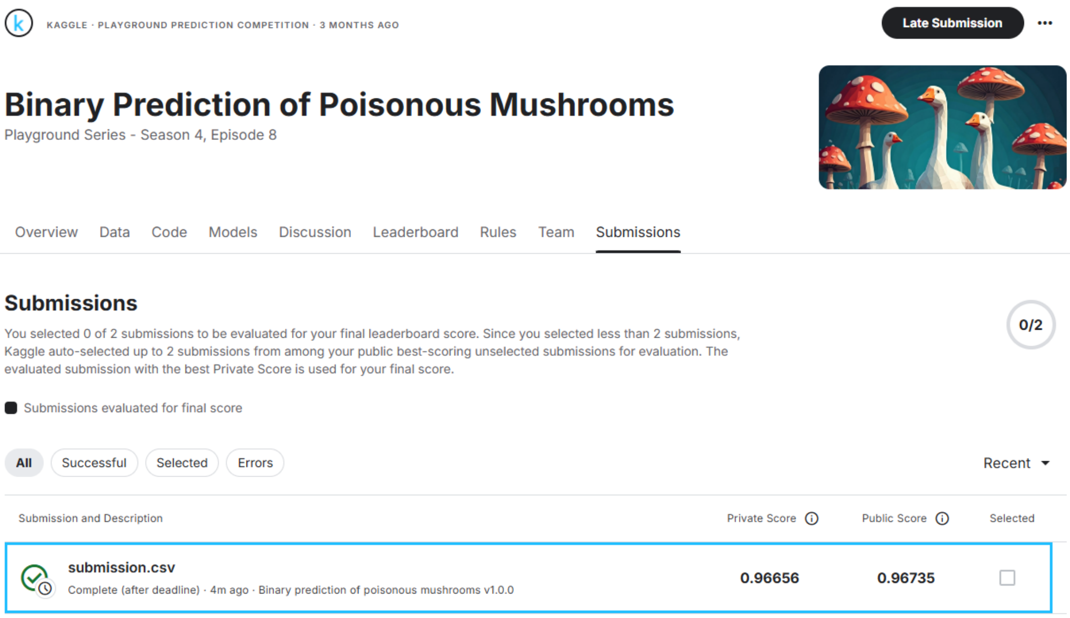
Code display (position: 05_model_prediction.py):

```
#prepare validate data
preprocess = Preprocess(test_set)
validateSet = ProcessedDataSet(preprocess.processed, feature_order=params['feature_order'])
validateSet.enc_onehot = params['enc_onehot']
validateSet.enc_minmax = params['enc_minmax']
validateSet()
print(validateSet.sample.head(5))
print(validateSet.sample.shape)
print(validateSet[0])
print(validateSet.X.shape)
[2]
```

We use the same encoder as in data preprocessing to reindex the test data.

Model prediction:

```
model_path = "../models/BinaryClassificationModel.pth"
output_file = "../data/processed/submission.csv"
batch_size = 32
input_dim = validateSet.sample.shape[1]
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")
✓ [3] 40毫秒

 Using device: cuda

import pandas as pd
from src.predict import Prediction
from torch.utils.data import DataLoader

predictor = Prediction(validateSet.X,model_path)
test_loader = DataLoader(validateSet, batch_size=batch_size, shuffle=False)
model = predictor.load_model().to(device)
predictions = predictor.predict(model, test_loader, device)
ids = validateSet.sample['id'].values
submission = pd.DataFrame({
    'id': ids,
    'class': ['p' if p == 1 else 'e' for p in predictions]
})
✓ [4] 59秒 94毫秒

submission.to_csv(output_file, index=False)
print(f"Predictions saved to {output_file}")
✓ [5] 1秒 747毫秒
 Predictions saved to ../data/processed/submission.csv
```



Kaggle score: 0.96735

VI. Appendix

Extra modules:

1. sys, os: applied to define the project root
2. sklearn: applied to do preprocessing, such as encoding labels, one-hot encoding categories, minmax encoding numerical data. Also used to do train-test set split (80% - 20%) and show metrics (accuracy score, precision score, recall score, f1_score).
3. pickle: applied to save processed data to avoid reprocessing.
4. matplotlib: applied to show figures

GitHub link: https://github.com/ZS912719/poisonous_mushrooms.git