数据挖掘与最优化: Assignment 3

组长:	231502004 李子	完成内容	T3	贡献度	20%
组员:	221300066 季千焜	完成内容	T4	贡献度	20%
组员:	221098024 李瑛琦	完成内容	T4	贡献度	20%
组员:	221098145 李傲雪	完成内容	T2	贡献度	20%
组员:	221098071 单佳仪	完成内容	T1	贡献度	20%

目录

实验进度

我们完成了所有内容。

T1

(i) 将样本随机分成两份，训练集样本量 1000，测试集样本量 1000

```
import torch
import numpy as np
import matplotlib.pyplot as plt
import torch.nn.functional as F

torch.manual_seed(123)
nobs = 1000
x = torch.linspace(-10,10,nobs)
y_train = 2 + torch.cos(2*x) + 0.5*torch.randn(nobs)
y_test = 2 + torch.cos(2*x) + 0.5*torch.randn(nobs)

x = x.reshape(nobs,1)
y_train = y_train.reshape(nobs,1)
y_test = y_test.reshape(nobs,1)
```

(ii) 基于训练集，绘制 (x,y) 散点图

```
plt.scatter(x.detach().numpy(), y_train.detach().numpy())
plt.show()
```

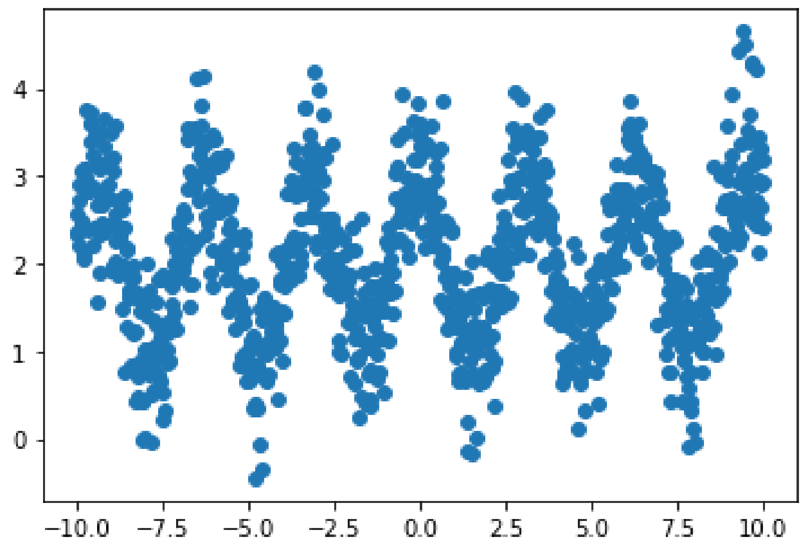

图 1: (x,y) 散点图

(iii) 基于训练集，使用双隐藏层（第一个隐藏层 20 个神经元，第二个隐藏层 10 个神经元）的神经网络对数据进行拟合，训练 300 步，每 30 步绘制训练效果图

```
# 定义神经网络:双隐藏层 (一层20, 二层10个神经元)
net = torch.nn.Sequential(
    torch.nn.Linear(1, 20),
    torch.nn.ReLU(),
    torch.nn.Linear(20, 10),
    torch.nn.ReLU(),
    torch.nn.Linear(10, 1)
)
print(net)

# 优化神经网络SGD(net.参数,    学习率=0.2)
# 均方差损失函数处理回归问题
optimizer = torch.optim.SGD(net.parameters(), lr=0.01)
loss_func = torch.nn.MSELoss()

# 训练300步
for t in range(300):
    prediction = net(x)      # input x and predict based on x
    loss = loss_func(prediction, y_train)      #loss_func(预测值, 真实值)

    optimizer.zero_grad()    # 使得梯度为0
    loss.backward()          # 反馈神经网络
    optimizer.step()         # 优化梯度
    #下面是可视化的过程
    if t % 30 == 0:#每30步打印过程
        # plot and show learning process
        plt.cla()
        plt.scatter(x.detach().numpy(), y_train.detach().numpy())
        plt.plot(x.detach().numpy(), prediction.detach().numpy(), 'r-', lw=5)
        plt.text(0.5, 0, 'Loss=%.4f' % loss.detach().numpy(), fontdict={'size': 20, 'color':  '
    red'})
        plt.pause(0.1)

plt.show()
```

训练效果图绘制如下：

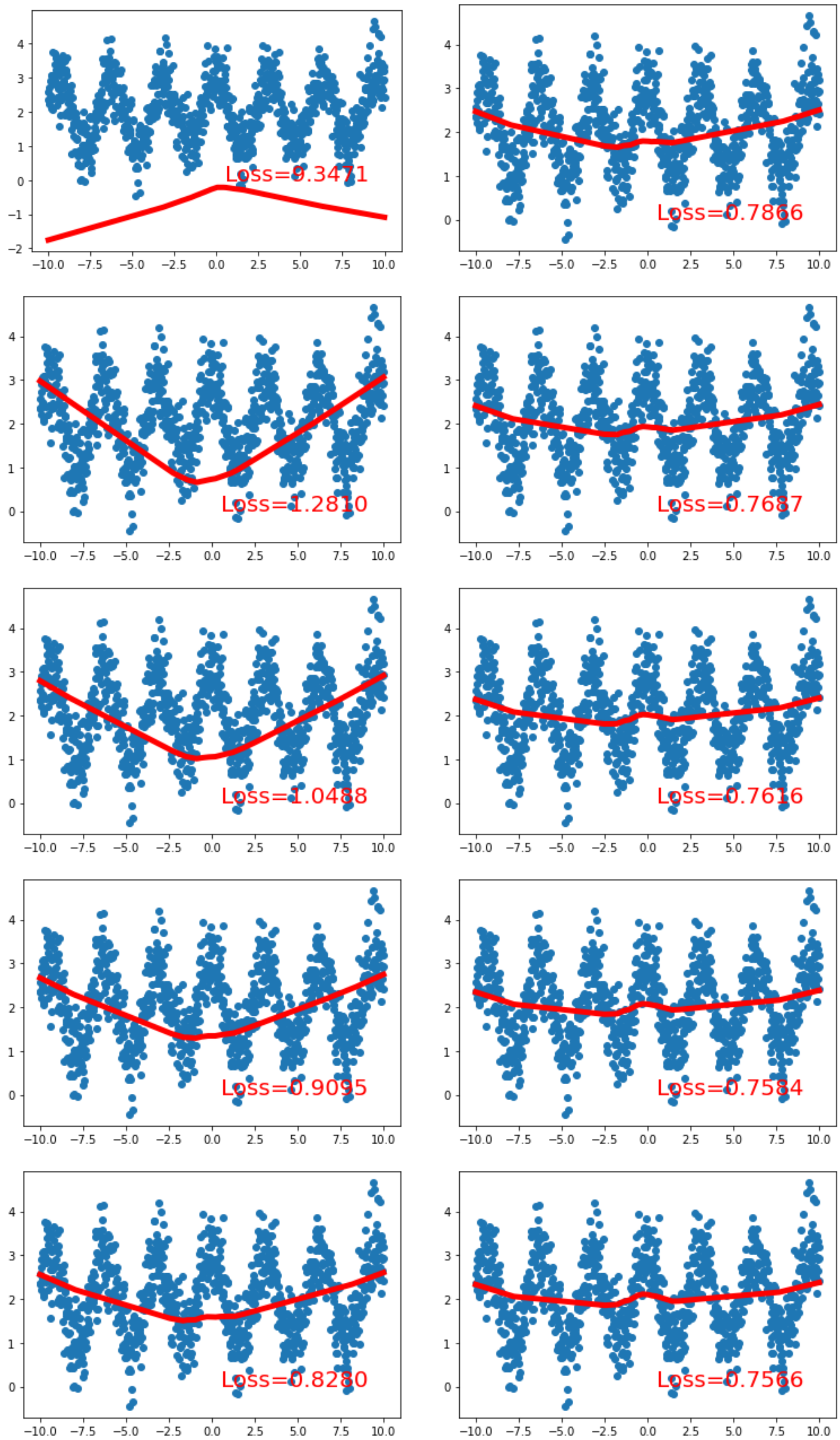

图 2: 训练效果图（左列为 1-5，右列为 6-10）

（iv）基于测试集和均方误差损失函数，评估神经网络的样本外预测表现

```python
with torch.no_grad():  # 关闭梯度计算以节省内存
    # 在测试集上进行预测
    y_pred_test = net(x)

    # 计算均方误差(MSE)
    mse_loss = loss_func(y_pred_test, y_test)

    # 计算R平方(决定系数)
    ss_tot = torch.sum((y_test - torch.mean(y_test))**2)
    ss_res = torch.sum((y_test - y_pred_test)**2)
    r_squared = 1 - (ss_res / ss_tot)

# 打印评估指标
print(f"\n测试集评估结果:")
print(f"MSE损失: {mse_loss.item():.4f}")
print(f"R平方值: {r_squared.item():.4f}")
```

测试集评估结果：MSE 损失：0.7719;R 平方值：0.0420
神经网络的样本外预测表现不是很好，建议通过增加训练次数等方法进行改进。

T2

我们完成了所有内容:

```python
import torch
import jieba

# ========== 数据预处理部分  ==========
# 中文原始文本(无需提前分词)
raw_text = """人生的意义在于不断追求进步和提升。
困难的环境是人生最好的教科书。
为明天做准备的最好方法就是今天做到最好。
伟大人物之所以伟大是因为他们决心成为伟大的人。
一次只做一件事是完成许多事情的最短路径。
只有在日常事务中尽责的人才能在重大场合尽责。
我全神贯注处理日常生活。
我能重新自立站起。
永远不要低估你改变自我的能力。"""

# 使用结巴自动分词
text = []
for sentence in raw_text.split('\n'):
    # 精确模式分词，关闭新词发现(提高切分准确性)
    words = jieba.cut(sentence, cut_all=False)
    text.extend([w for w in words if w.strip()])  # 去除空字符

print("分词结果示例:", text[:10])  # 打印前10个分词结果

# ========== 构建词汇表  ==========
word = set(text)
word_size = len(word)
print("\n词汇表大小:", word_size)
print("示例词汇:", list(word)[:5])  # 打印前5个词

# 建立词到索引的映射
word_to_ix = {word:ix for ix, word in enumerate(word)}
ix_to_word = {ix:word for ix, word in enumerate(word)}

# ========== 模型架构部分  ==========
class CBOW(torch.nn.Module):
    def __init__(self, vocab_size, emb_dim=100, h1_dim=200, h2_dim=200):
        super().__init__()
        self.embeddings = torch.nn.Embedding(vocab_size, emb_dim)
        self.hidden1 = torch.nn.Linear(emb_dim, h1_dim, bias=True)
```

```
        self.hidden2 = torch.nn.Linear(h1_dim, h2_dim, bias=True)
        self.output = torch.nn.Linear(h2_dim, vocab_size, bias=True)
        self.relu = torch.nn.ReLU()

    def forward(self, context):
        embeds = torch.mean(self.embeddings(context), dim=0)
        out = self.relu(self.hidden1(embeds))
        out = self.relu(self.hidden2(out))
        return torch.softmax(self.output(out), dim=0)

# ========== 训练配置 ==========
model = CBOW(len(word), emb_dim=100, h1_dim=200, h2_dim=150)
print("\n模型结构:")
print(model)

# 创建训练数据 (窗口大小为2)
data = []
for i in range(2, len(text)-2):
    context = [text[i-2], text[i-1], text[i+1], text[i+2]]  # 上下文窗口
    target = text[i]  # 目标词
    data.append((context, target))

# 训练参数
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)  # 改用Adam优化器

# ========== 训练循环 ==========
print("\n开始训练...")
for epoch in range(300):
    total_loss = 0
    for context, target in data:
        # 转换上下文为索引
        context_idx = torch.tensor([word_to_ix[w] for w in context], dtype=torch.long)

        # 前向传播
        probs = model(context_idx)
        target_idx = torch.tensor([word_to_ix[target]], dtype=torch.long)

        # 计算损失
        loss = loss_fn(probs.unsqueeze(0), target_idx)

        # 反向传播
        optimizer.zero_grad()
        loss.backward()
```

```
            optimizer.step()

            total_loss += loss.item()

        if epoch % 30 == 0:
            print(f"Epoch {epoch:03d} | Loss {total_loss/len(data):.4f}")

    # ========== 词向量提取 ==========
    print("\n提取词向量...")

    # 获取词向量矩阵
    embeddings = model.embeddings.weight.data.numpy()

    print(f"词向量矩阵形状: {embeddings.shape}")
    print(f"词汇表大小: {embeddings.shape[0]}")
    print(f"词向量维度: {embeddings.shape[1]}")

    print("\n完整词向量矩阵:")
    print("=" * 50)
    print(embeddings)
    print("=" * 50)

    # 词汇到索引的对应关系
    # print("\n词汇索引对应关系:")
    # for word, idx in sorted(word_to_ix.items(), key=lambda x: x[1]):
    #     print(f"索引 {idx:2d}: {word}")

    print(f"\n词向量矩阵提取完成! ")
```

分词结果与词汇表

- **分词结果示例**: ['人生', '的', '意义', '在于', '不断', '追求进步', '和', '提升', '。', '困难']
- **词汇表信息**:
  - 词汇表大小: 65
  - 示例词汇: ['环境', '事情', '他们', '人', '许多']

模型结构

- CBOW 模型结构:

- `embeddings`: Embedding(65, 100)
- `hidden1`: Linear(in_features=100, out_features=200, bias=True)
- `hidden2`: Linear(in_features=200, out_features=150, bias=True)
- `output`: Linear(in_features=150, out_features=65, bias=True)
- `relu`: ReLU()

训练过程

Epoch	**Loss**
Epoch 000	4.1747
Epoch 030	3.3213
Epoch 060	3.2952
Epoch 090	3.4874
Epoch 120	3.2833
Epoch 150	3.2718
Epoch 180	3.2719
Epoch 210	3.2837
Epoch 240	3.2838
Epoch 270	3.2838

词向量矩阵

词向量矩阵形状：(65，100)，词汇表大小：65，词向量维度：100

$$\mathbf{W} = \begin{bmatrix} -0.6927251 & 1.550888 & 0.39339858 & \cdots & 0.46651623 & -0.59766465 & 0.8609725 \\ -0.7312393 & 0.79728645 & 1.611781 & \cdots & -0.18673758 & -1.4100798 & -0.55437416 \\ 0.42163095 & 2.2570329 & -0.05199318 & \cdots & 0.13223904 & -0.05400054 & -1.090873 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ -0.6403143 & -0.72123086 & 1.3191246 & \cdots & 1.3094645 & 0.5098204 & 1.0451326 \\ -1.5902704 & -0.32260102 & 1.2048887 & \cdots & -0.4586594 & 1.0261565 & 1.4510612 \\ 0.13485973 & 0.08004124 & -0.2552207 & \cdots & 0.7548248 & -0.79750603 & 0.6621087 \end{bmatrix}$$

T3

训练代码如下：

```python

import jieba
import logging
from gensim.models import Word2Vec
import re
from gensim.models.phrases import Phrases, Phraser
import jieba.posseg as pseg

logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s', level=logging.INFO)

jieba.add_word('张无忌')
jieba.add_word('冰火岛')
jieba.add_word('武当派')
jieba.add_word('九阳神功')
jieba.add_word('乾坤大挪移')

def preprocess_text(file_path):

    custom_entities = [
        '张无忌', '赵敏', '周芷若', '谢逊', '武当派', '明教',
        '冰火岛', '屠龙刀', '倚天剑', '九阳神功', '乾坤大挪移',
        '张三丰', '殷素素', '张翠山', '小昭', '殷离', '汝阳王府',
        '紫衫龙王', '金毛狮王', '光明顶', '波斯明教'
    ]
    for word in custom_entities:
        jieba.add_word(word, freq=1000, tag='n')

    jieba.del_word('张无')
    jieba.del_word('冰火')
    jieba.del_word('两派')
    jieba.del_word('武当')
    for encoding in ['utf-8', 'gbk', 'gb18030']:
        try:
            with open(file_path, 'r', encoding=encoding) as f:
                text = f.read()
            break
        except UnicodeDecodeError:
            continue

    text = text.replace('\u3000', ' ').replace('\n', ' ').replace('\r', ' ')
```

```

    paragraphs = [p.strip() for p in text.split('。') if len(p) > 10]
    stopwords = set(['的', '了', '在', '是', '我', '你', '他', '这', '那', '就'])

    # 分词处理
    sentences = []
    for para in paragraphs:
        para = re.sub(r'[^\u4e00-\u9fa5a-zA-Z0-9]', ' ', para)
        para = re.sub(r'\s+', ' ', para)  #
        words = list(jieba.cut(para))
        sentences.append([w for w in words if
                          (w not in stopwords) and
                          (len(w) > 1 or w in ['张', '谢'])])
    phrases = Phrases(sentences, min_count=5, threshold=10)
    bigram = Phraser(phrases)
    sentences = [bigram[s] for s in sentences]

    filtered = []
    for sent in sentences:
        words = pseg.cut(' '.join(sent))
        filtered.append([w.word for w in words if w.flag.startswith('n') and len(w.word)>1])
    return filtered

def train_word2vec(sentences, model_save_path):
    model = Word2Vec(
        sentences=sentences,
        vector_size=200,
        window=8,
        min_count=2,
        sg=1,
        hs=0,
        negative=5,
        sample=1e-4,
        alpha=0.025,
        min_alpha=0.0007,
        epochs=100
    )

    model.save(model_save_path)
    return model

def main():
    # 配置文件路径
    novel_path = "倚天屠龙记.txt"
```

```python
    model_path = "yitian_w2v.model"

    # 预处理
    print("正在预处理文本...")
    sentences = preprocess_text(novel_path)

    # 检查数据样例
    print("\n分词示例：")
    print(sentences[10][:10])

    # 训练模型
    print("\n开始训练模型...")
    model = train_word2vec(sentences, model_path)

    # 验证训练效果
    test_words = ["张无忌", "武当派", "冰火岛", "屠龙刀", "赵敏"]
    for word in test_words:
        if word in model.wv:
            print(f"\n{word} 的相似词：")
            print([(w, round(s, 3)) for w, s in model.wv.most_similar(word, topn=5)])
        else:
            print(f"\n警告：'{word}' 不在词汇表中（出现次数不足）")

if __name__ == "__main__":
    main()
```

输出：

```
张无忌 的相似词：
[('赵敏', 0.377), ('众人', 0.357), ('小昭', 0.356), ('赵姑娘', 0.355), ('灭绝师太', 0.342)]

武当派 的相似词：
[('孟正鸿', 0.415), ('江湖', 0.412), ('五侠', 0.391), ('椅子', 0.379), ('张三丰', 0.377)]

冰火岛 的相似词：
[('波涛', 0.495), ('故事', 0.471), ('极北', 0.45), ('朱长龄', 0.443), ('伯伯', 0.434)]

屠龙刀 的相似词：
[('宝刀', 0.422), ('秘笈', 0.413), ('倚天剑', 0.41), ('狂性', 0.379), ('铁锤', 0.371)]

赵敏 的相似词：
[('赵姑娘', 0.479), ('周芷若', 0.384), ('张无忌', 0.377), ('小昭', 0.376), ('姑娘', 0.355)]
```

T4

将一篇影评中所有词对应的词向量平均作为该影评的向量表示如下:

```
import numpy as np
import pandas as pd
from bs4 import BeautifulSoup
import re
import nltk
from nltk.corpus import stopwords
data=pd.read_csv(r"labeledTrainData.tsv",header=0,delimiter="\t",nrows=1000)
data
# 下载英文停用词
nltk.download("stopwords",download_dir=r"E:/nltk_data")
# 英文文本预处理
def review_to_words(raw_review):
    review_text = BeautifulSoup(raw_review).get_text() #原始的评论文本中有大量的HTML标记符号，使用BeautifulSoup的解析库对其进行清理
    letters_only = re.sub("[^a-zA-Z]", " ", review_text) #通过正则表达式清除掉一些标点等非字母字符
    words = letters_only.lower().split()
    stops = set(stopwords.words("english"))
    meaningful_words = [w for w in words if w not in stops]
    return( " ".join( meaningful_words))
data["clean_review"]=data["review"].apply(review_to_words)
data
# 将影评中出现单词形成一个集合
word_list = set(data.clean_review.str.split().explode().tolist())
# 加载100维腾讯词向量
def load_tencent_vectors(file_path, word_list,embed_size):
    word2vec = {}
    with open(file_path, "r",encoding="utf-8") as f:
        for line_num,line in enumerate(f):
            parts = line.strip().split()
            word = parts[0]
            if word not in word_list:
                continue
            vector = np.array([float(x) for x in parts[1:]])
            word2vec[word] = vector
    print(f"成功加载 {len(word2vec)} 个词向量 (维度: {embed_size}) ")
    return word2vec
word2vec = load_tencent_vectors("glove.6B.200d.txt",word_list,embed_size=200)
word2vec
# 将一篇影评表示为对应的100维腾讯词向量平均
# 对词向量进行归一化
```

```python
from sklearn.preprocessing import normalize

def review_to_vector(line, word2vec, embed_size):
    vec = np.zeros(embed_size, dtype=np.float32)  # 初始化零向量
    count = 0
    for word in line:
        if word in word2vec:
            vec += normalize(word2vec[word].reshape(1, -1))  # 归一化词向量
            count += 1
    if count > 0:
        vec /= count  # 求平均
    return pd.Series(vec)
# 代入1000条影评将影评中所有词对应的词向量平均作为该影评的向量表示
embed_size = 200
clean_review_list = data.clean_review.str.split()
train_data_features = pd.DataFrame([review_to_vector(line,word2vec,embed_size) for line in
clean_review_list])
train_data_features
# 划分训练集与测试集
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(train_data_features,data.sentiment,test_size
=0.2,random_state=0)
#Logistic regression
from sklearn.linear_model import LogisticRegression
LR_model=LogisticRegression()
LR_model=LR_model.fit(X_train,y_train)
y_pred=LR_model.predict(X_test)
# Confusion matrix
from sklearn.metrics import confusion_matrix
cnf_matrix=confusion_matrix(y_test,y_pred)

print("Accuracy is: ",(cnf_matrix[0,0]+cnf_matrix[1,1])/(cnf_matrix[0,0]+cnf_matrix[1,1]+
cnf_matrix[0,1]+cnf_matrix[1,0]))
print("Sensitivity is: ",cnf_matrix[1,1]/(cnf_matrix[1,1]+cnf_matrix[1,0]))
print("Specificity is: ",cnf_matrix[0,0]/(cnf_matrix[0,0]+cnf_matrix[0,1]))

import matplotlib.pyplot as plt
import itertools

def plot_confusion_matrix(cm,classes,title="Confusion matrix",cmap=plt.cm.Blues):
    plt.imshow(cm,interpolation="nearest",cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks=np.arange(len(classes))
```

```python
    plt.xticks(tick_marks,classes,rotation=0)
    plt.yticks(tick_marks,classes)

    thresh=cm.max()/2
    for i,j in itertools.product(range(cm.shape[0]),range(cm.shape[1])):
        plt.text(j,i,cm[i,j],horizontalalignment="center",color="white" if cm[i,j]>thresh else "black")

    plt.tight_layout()
    plt.ylabel("True label")
    plt.xlabel("Predicted label")

class_names=[0,1]
plt.figure()
plot_confusion_matrix(cnf_matrix,classes=class_names)
plt.show()
```

输出结果如下:

Accuracy is: 0.795

Sensitivity is: 0.8350515463917526

Specificity is: 0.7572815533980582