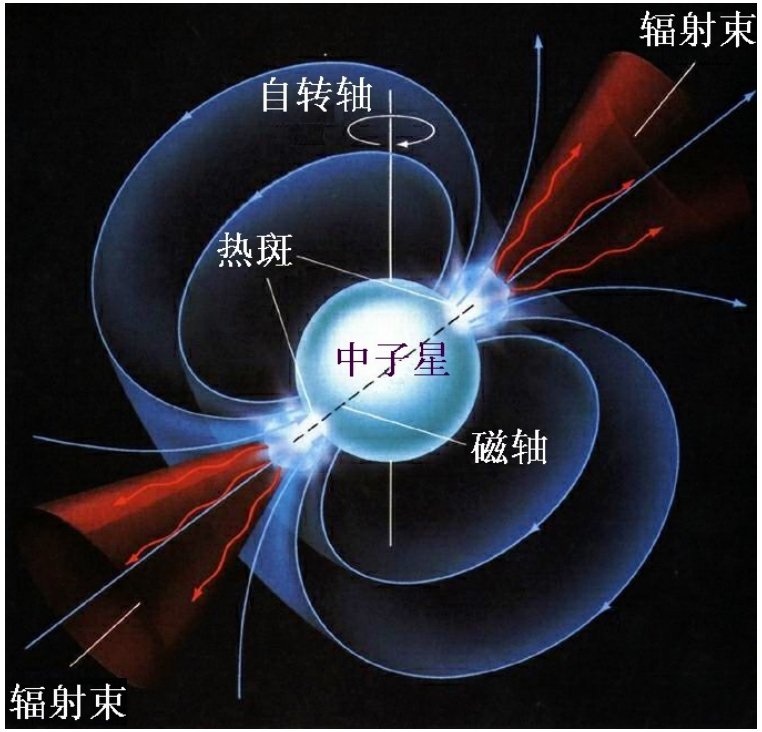
脉冲星分类方法

脉冲星简介



脉冲星是一种稀有的超级致密的中子星，每当脉冲星极冠转到地球视线方向，我们便看到其辐射信号，所以感觉收到了脉冲信号，脉冲星好比航海的灯塔，当辐射束扫过地球即可以观测一次脉冲。脉冲星半径约10 千米、自旋很快，目前在射电波段观测到的旋转周期约在1.4毫秒~8.5 秒之间。

作为时空、星际介质和物质状态的探测器，它们具有相当大的科学意义：

1）由于脉冲星的大质量和小半径，其表面引力场非常强，所以不能忽略广义相对论效应的存在，这使得脉冲星成为强引力场研究的天然实验室。

2）由于脉冲星的超强磁场，这为我们研究磁层粒子加速机制、高能辐射、射电辐射过程提供了一个理想场所。

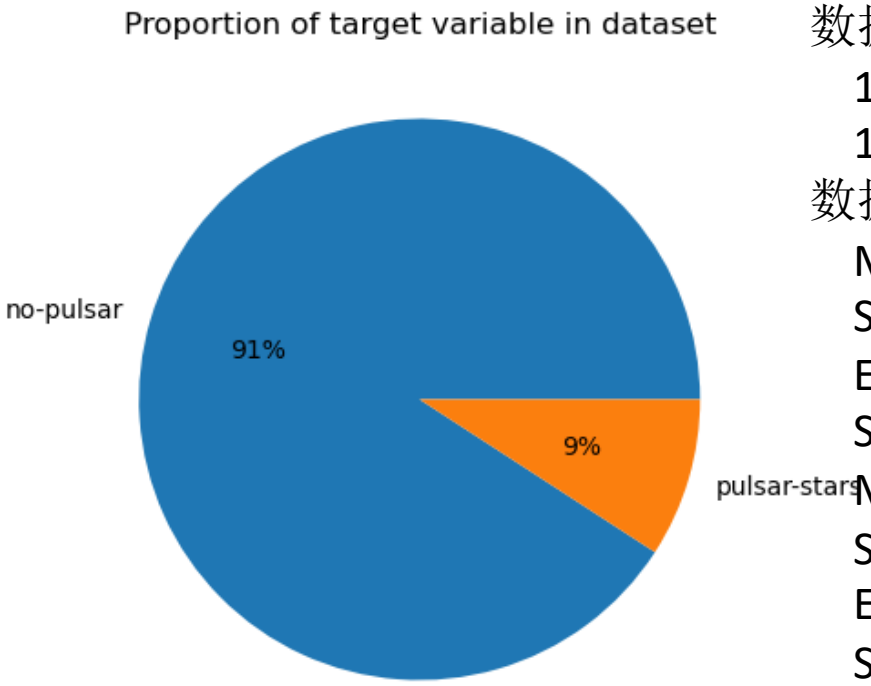
3）脉冲星作为大质量恒星坍缩后超新星爆发的产物，它对于研究超新星爆发理论、理解脉冲星的形成机制相当重要。

4）在应用研究方面，脉冲星因其自转周期的高度稳定性，在时间标准和航天器导航上有非常重要的应用前景。

天文学家已经观测到2700多颗脉冲星，至今已积累了一大批宝贵的资料，同时也存在不少现象和问题尚待解决。

自2017年10月10日首次对外宣布发现脉冲星以来，截至目前，被誉为“中国天眼”的500米口径球面射电望远镜已发现660余颗新脉冲星。

数据集介绍



数据集概况：

16259条因射频干扰而像脉冲星的记录，

1639条真真正的脉冲星记录

数据属性：

Mean of the integrated profile : 脉冲轮廓宽度的平均值

Standard deviation of the integrated profile: 脉冲轮廓宽度的标准差

Excess kurtosis of the integrated profile : 脉冲轮廓宽度高于阈值的脉冲占比

Skewness of the integrated profile : 脉冲轮廓宽度偏度

Mean of the DM-SNR curve : DM-SNR(色散度-信噪比)曲线均值

Standard deviation of the DM-SNR curve : DM-SNR曲线标准差

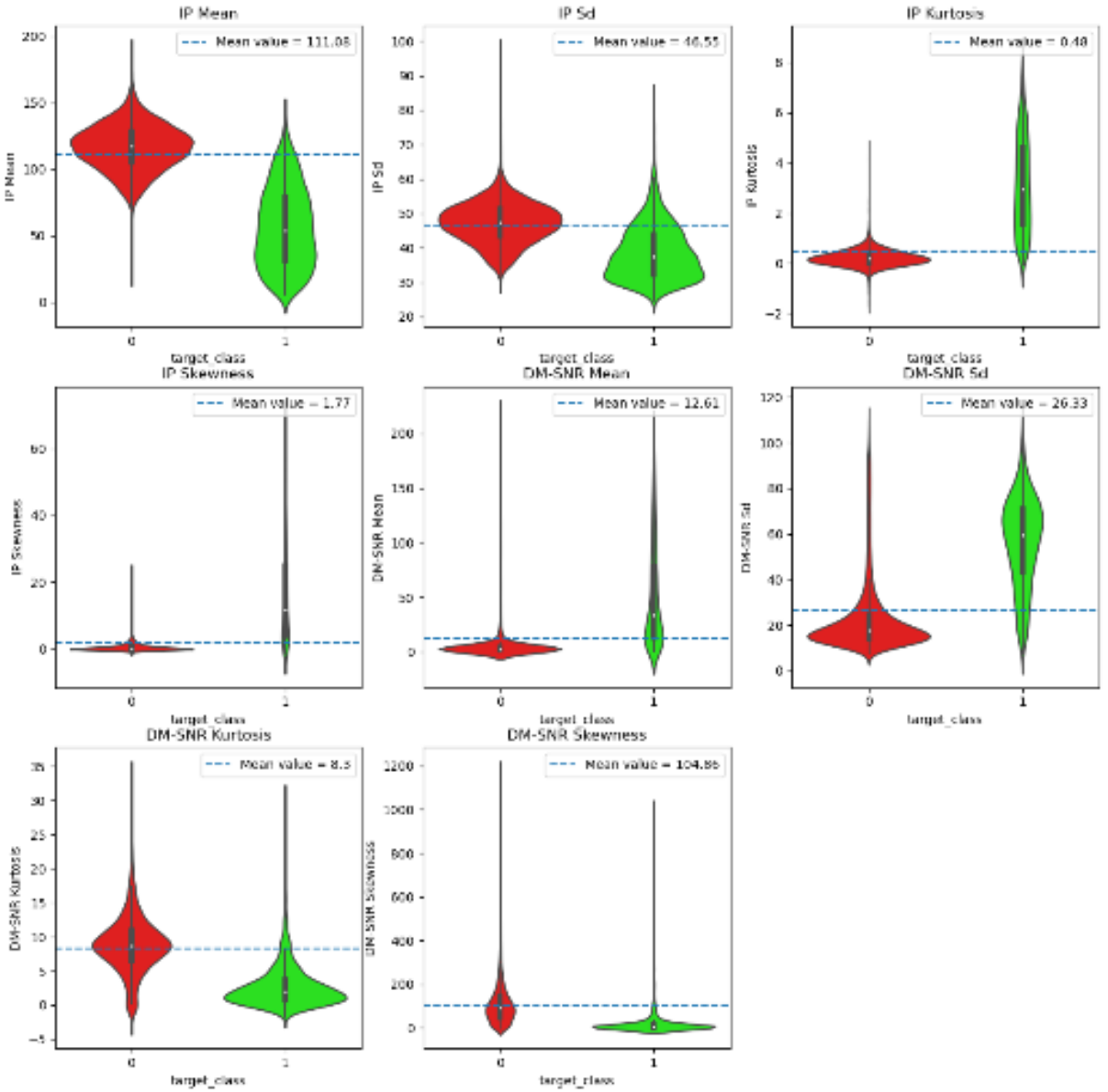
Excess kurtosis of the DM-SNR curve : DM-SNR曲线的超额峰度

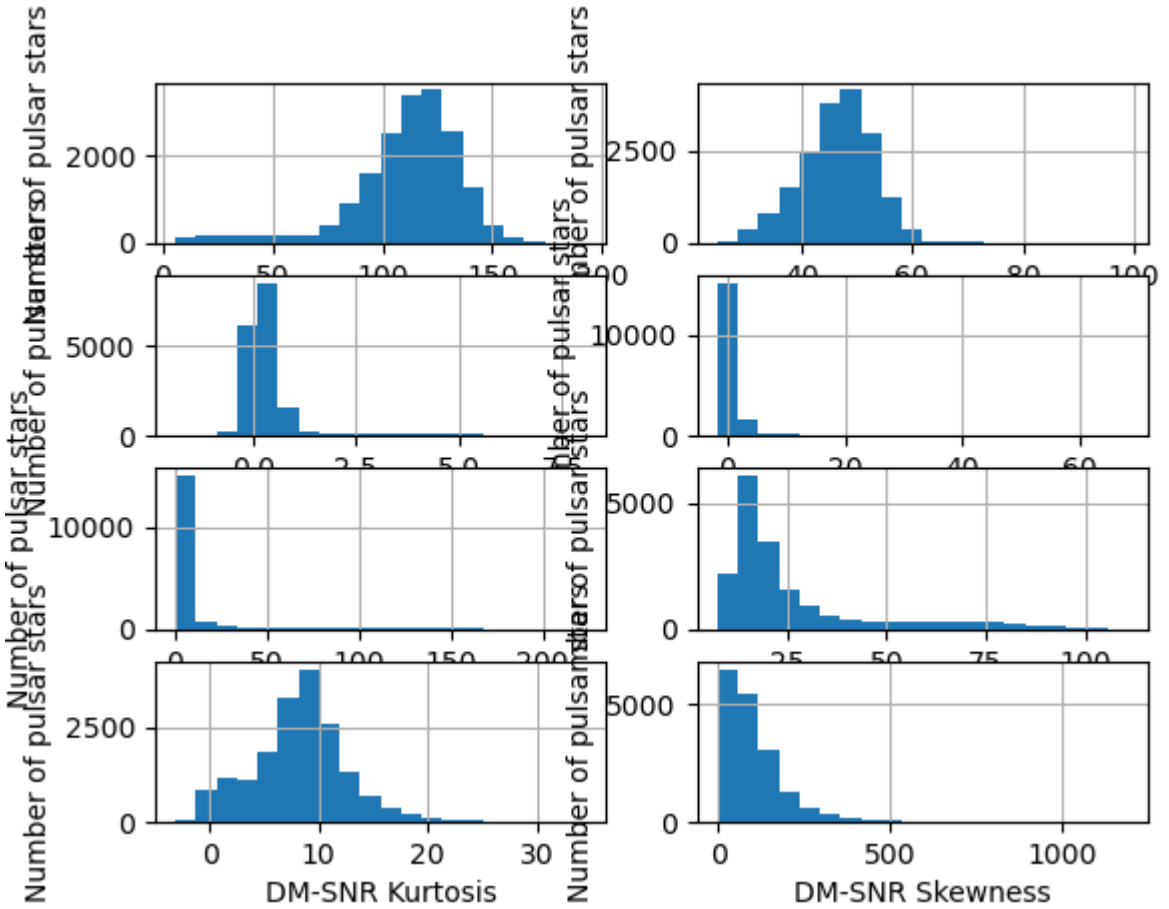
Skewness of the DM-SNR curve : DM-SNR曲线偏度

Class : 目标类型(label)

数据集地址：https://archive.ics.uci.edu/ml/datasets/HTRU2

数据分布


IP Mean
Mean value = 111.08
IP Sd
Mean value = 46.55
IP Kurtosis
Mean value = 0.48
IP Skewness
Mean value = 1.77
DM-SNR Mean
Mean value = 12.61
DM-SNR Sd
Mean value = 26.33
DM-SNR Kurtosis
Mean value = 8.3
DM-SNR Skewness
Mean value = 104.86
target_class


Number of pulsar stars
DM-SNR Kurtosis
DM-SNR Skewness

数据集预处理

```
# 数据预处理
def date_preprocess(df: pd.DataFrame):
    # 区分特征向量和目标值
    y = df.get('target_class')

    # 分割训练集为训练集(80%)和测试集(20%)
    x_train, x_test, y_train, y_test = \
        train_test_split(df, y, test_size=0.2, random_state=3)

    # 下采样
    train_1 = x_train[x_train['target_class'] == 1]
    train_0 = x_train[x_train['target_class'] == 0]
    sample = train_0.sample(train_1.shape[0])
    train_all = train_1.append(sample)

    # 构造训练集 测试集
    svm.y_train = train_all.get('target_class')
    svm.x_train = train_all.drop(['target_class'], axis=1)
```

1）按80%：20% 的比例分割训练集和测试集
2）下采样的方式解决数据不平衡问题
3）特征缩放，将特征数据分布调整为标准正态分布，均值为0，方差为1

实验证明，进行数据下采样后再训练模型，测试结果都比较差，直接使用原始数据的效果反而不错。

SVM训练与测试

```
# C：惩罚系数，数值越高越容易过拟合
# gamma：是选择径向基函数（RBF）作为kernel后，该函数自带的一个参数
# gamma越大，支持向量越少，gamma越小，支持向量越多。数值越大越容易过拟合
def svm_train(kernel="rbf", C=1.0, gamma='auto'):
    svc = SVC(kernel=kernel, C=C, gamma=gamma)

    svc.fit(svm.x_train, svm.y_train)

    svm.y_pred_train = svc.predict(svm.x_train)
    train_score = accuracy_score(svm.y_train, svm.y_pred_train)

    # 预测
    svm.y_pred = svc.predict(svm.x_test)
    test_score = accuracy_score(svm.y_test, svm.y_pred)

    # print(score)

    print('Train Model accuracy score with kernel:{0}, C:{1}, gamma:{2} hyper parameters: {3:0.4f}'
          .format(kernel, C, gamma, train_score))
    print('Test Model accuracy score with kernel:{0}, C:{1}, gamma:{2} hyper parameters: {3:0.4f}'
          .format(kernel, C, gamma, test_score))
```

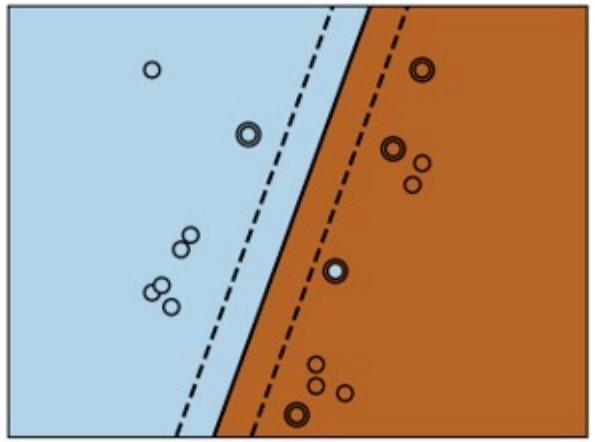
分数评价指标

```
# 模型指标评价
def performance():
    # 混淆矩阵
    class_names = np.array(['0', '1'])
    with parallel_backend('threading', n_jobs=8):
        cm = confusion_matrix(svm.y_test, svm.y_pred)
        plot_confusion_matrix(cm, class_names)

    TP = cm[0, 0]
    TN = cm[1, 1]
    FP = cm[0, 1]
    FN = cm[1, 0]
    # accuracy 精度 精度则是分类正确的样本数占样本总数的比例
    classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)
    # error 错误率 错误率是分类错误的样本数占样本总数的比例
    classification_error = (FP + FN) / float(TP + TN + FP + FN)
    # precision 查准率 正确预测的正样本数占所有预测为正样本的数量的比值
    precision = TP / float(TP + FP)
    # recall(true positive rate) 查全率/召回率 正确预测的正样本数占真实正样本总数的比值
    recall = TP / float(TP + FN)
    false_positive_rate = FP / float(FP + TN)
    specificity = TN / (TN + FP)
```

8线程

```
print('Co
print('Tr
print('Tr
print('Fa
print('Fa

# print(c
print('Cl                                                    cy))
```

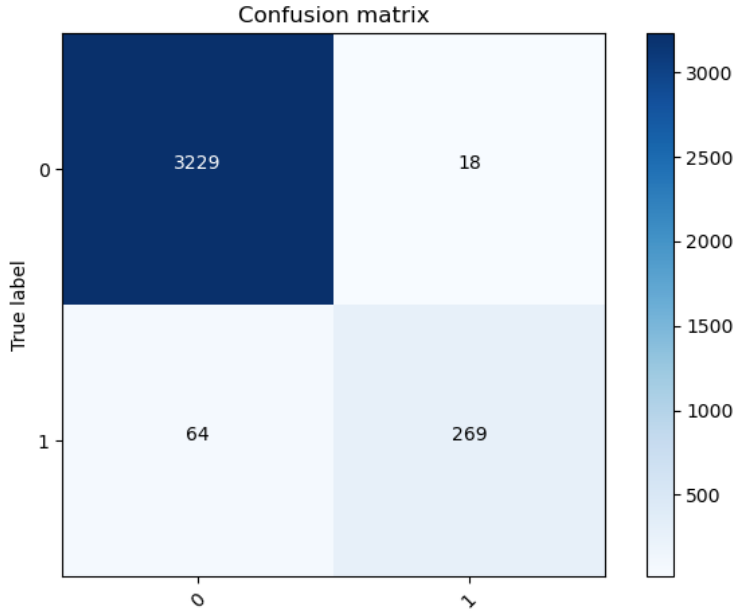
线性核函数

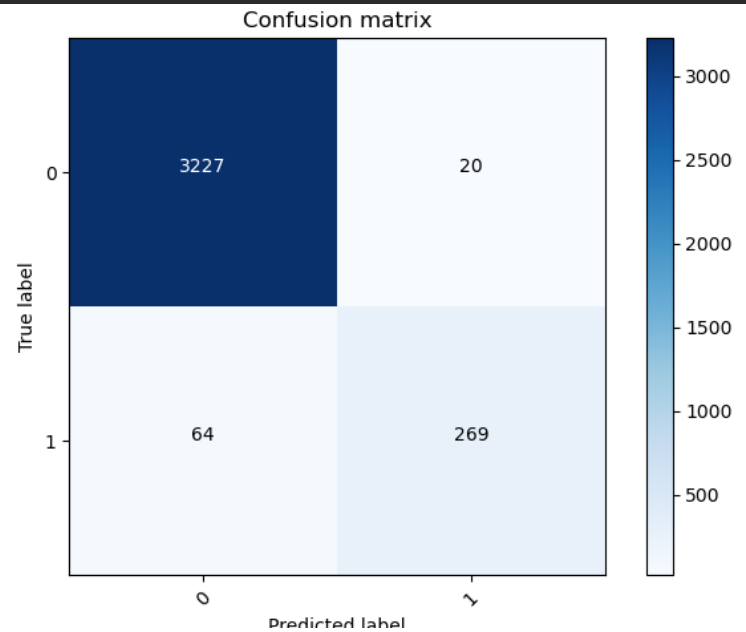


```
# 线性
def svm_linear():
    print("############## linear kernel ##############")

    kernel = 'linear'
    svm_train(kernel=kernel)

    svm_train(kernel=kernel, C=1.0)

    svm_train(kernel=kernel, C=100.0)

    svm_train(kernel=kernel, C=1000.0)
```

结果分析

```
############# linear kernel #############
kernel:linear, C:1.0, gamma:auto hyper parameters
Train Model accuracy score: 0.9804
Train Model accuracy score: 0.9771
Train Model cross-validation scores:
 [0.97835196 0.98219274 0.98218652 0.97589941 0.9804401 ]
Test Model cross-validation scores :
 [0.98603352 0.97765363 0.97206704 0.97346369 0.97346369]
Train Model Average stratified  cross-validation scores 0.9798
Test Model Average stratified cross-validation scores: 0.9765
True Positives(TP) =  3229
True Negatives(TN) =  269
False Positives(FP) =  18
```



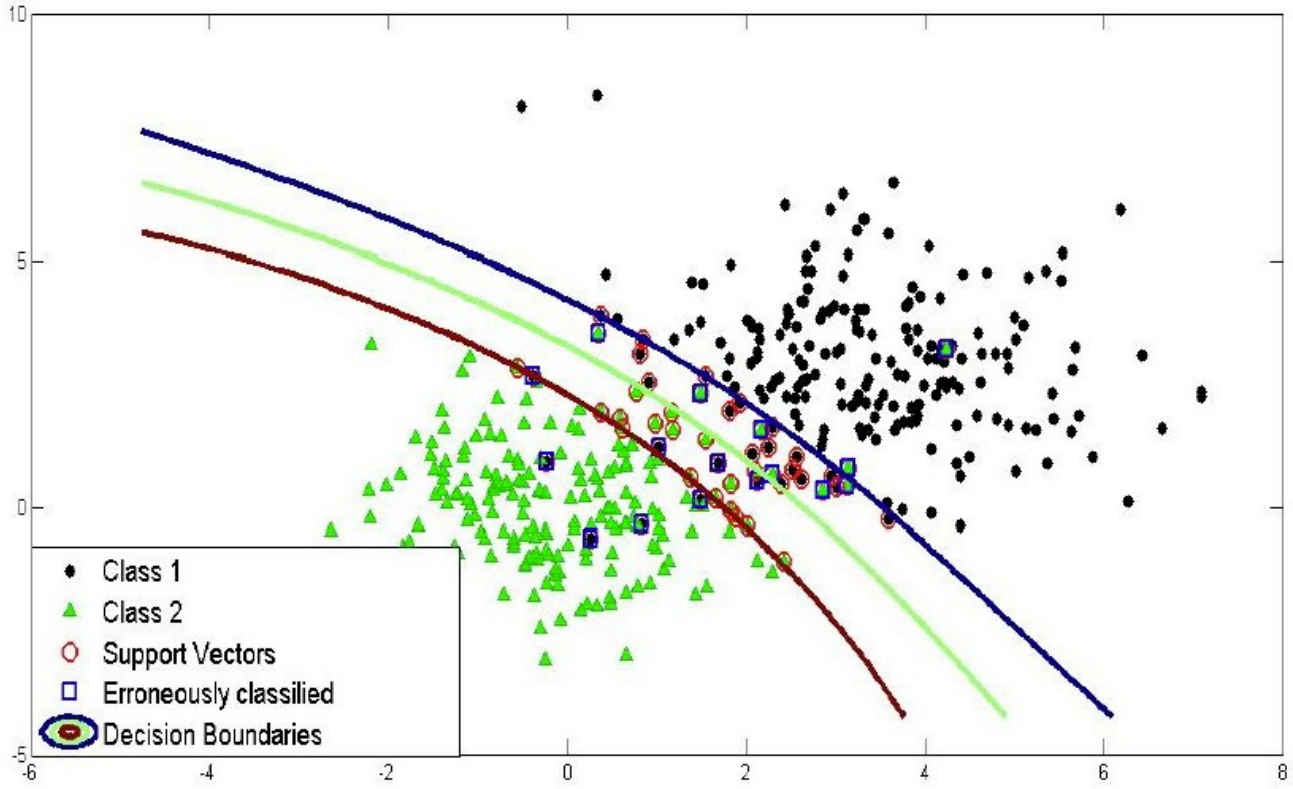



```
kernel:linear, C:100.0, gamma:auto hyper parameters
Train Model accuracy score: 0.9803
Train Model accuracy score: 0.9765
Train Model cross-validation scores:
 [0.97835196 0.9825419  0.98218652 0.97694726 0.98113867]
Test Model cross-validation scores :
 [0.98463687 0.97765363 0.97206704 0.97346369 0.97486034]
Train Model Average stratified  cross-validation scores 0.9802
Test Model Average stratified cross-validation scores: 0.9765
True Positives(TP) =  3227
True Negatives(TN) =  269
False Positives(FP) =  20
False Negatives(FN) =  64
```

RBF 径向基函数

Radial Basis Function kernel

$$k(x,y)=\exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$$

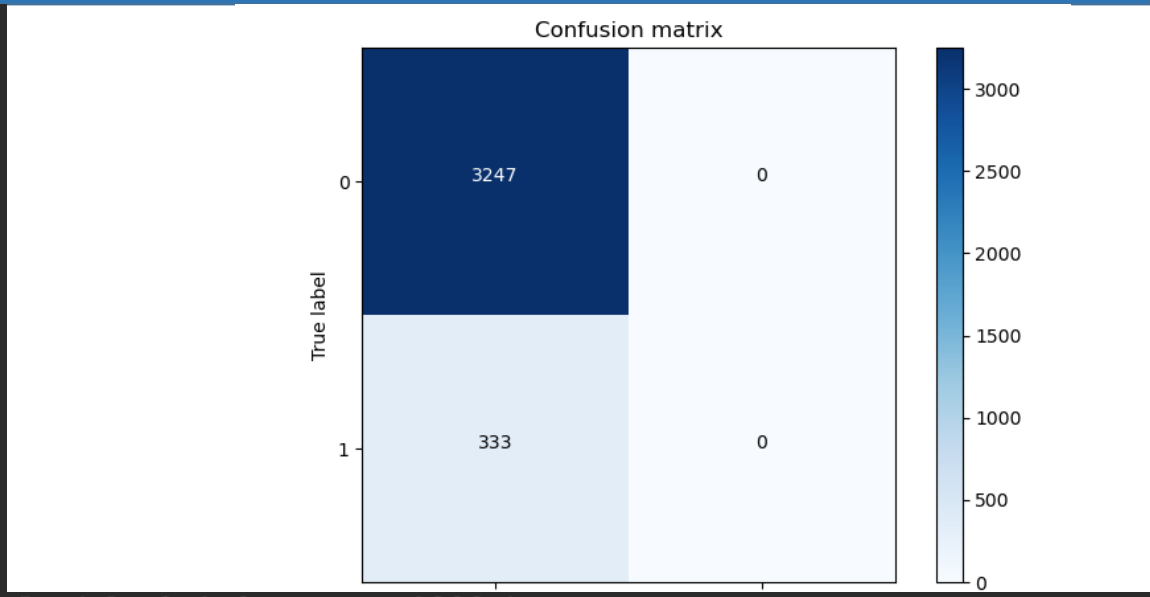


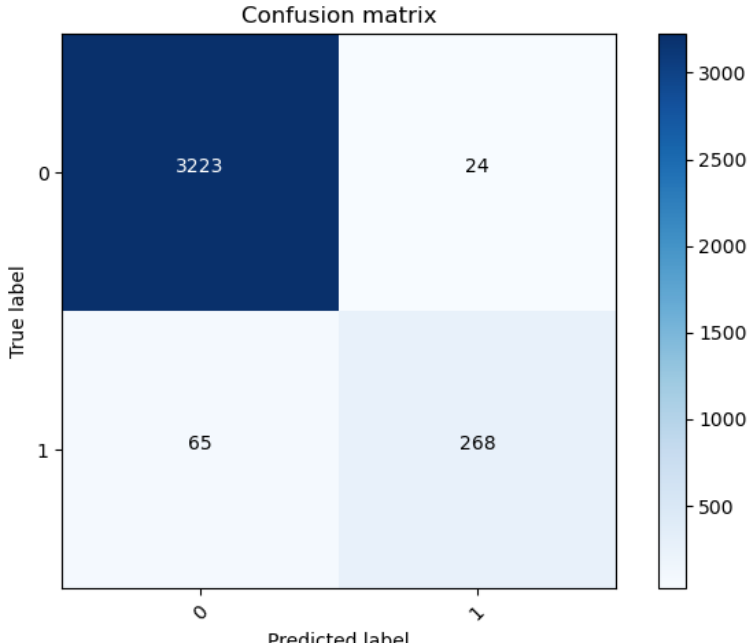

```python
# 径向核函数
def svm_rbf():
    print("############## rbf kernel ##############")

    kernel = 'rbf'
    svm_train(kernel=kernel)

    svm_train(kernel=kernel, gamma=0.1)

    svm_train(kernel=kernel, gamma=0.5)

    svm_train(kernel=kernel, gamma=0.9)

    svm_train(kernel=kernel, gamma=1)

    svm_train(kernel=kernel, gamma=100)

    svm_train(kernel=kernel, gamma=1000)
```

RBF可以作为一个通用的核函数使用，经常在我们对数据了解不多的时候使用，也是svm的默认核函数。

结果分析

```
############# rbf kernel #############
kernel:rbf, C:1.0, gamma:auto hyper parameters
Train Model accuracy score: 0.9802
Train Model accuracy score: 0.9751
Train Model cross-validation scores:
 [0.97590782 0.98114525 0.98218652 0.97555012 0.98078938]
Test Model cross-validation scores :
 [0.98044693 0.97625698 0.97067039 0.97346369 0.97206704]
Train Model Average stratified  cross-validation scores 0.9791
Test Model Average stratified cross-validation scores: 0.9746
```

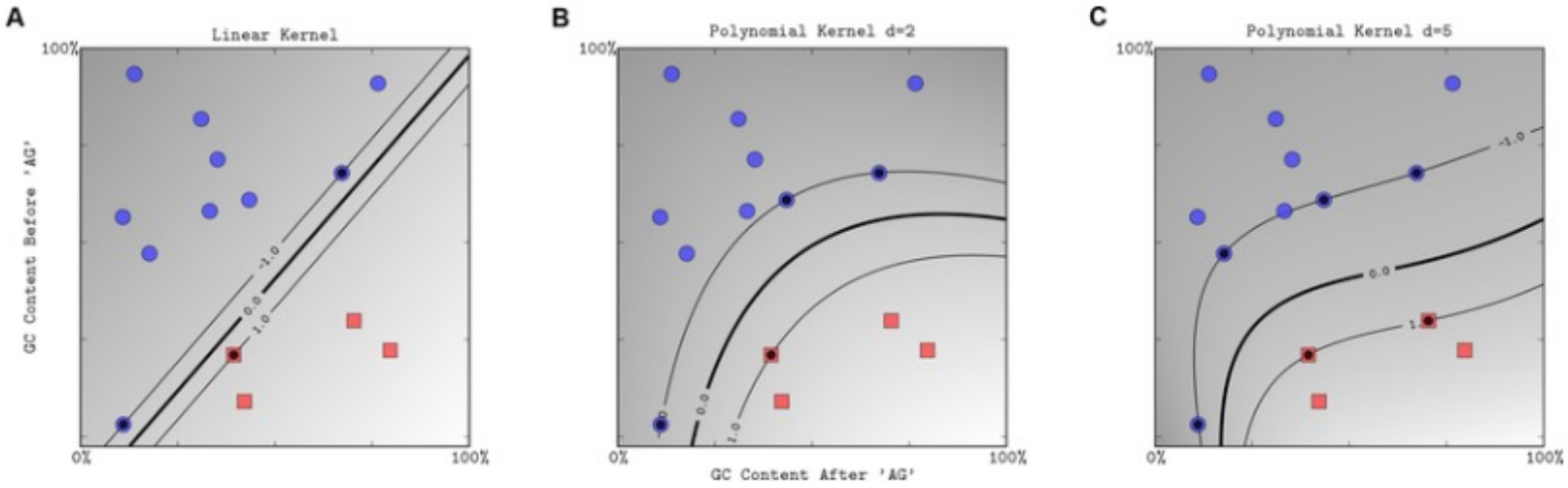





```
kernel:rbf, C:1.0, gamma:1000 hyper parameters
Train Model accuracy score: 1.0000
Train Model accuracy score: 0.9070
Train Model cross-validation scores:
 [0.9127095  0.90572626 0.91372686 0.89451624 0.9172197 ]
Test Model cross-validation scores :
 [0.89804469 0.92178771 0.91201117 0.89944134 0.90363128]
Train Model Average stratified  cross-validation scores 0.9088
Test Model Average stratified cross-validation scores: 0.9070
True Positives(TP) =  3247
True Negatives(TN) =  0
False Positives(FP) =  0
False Negatives(FN) =  333
```

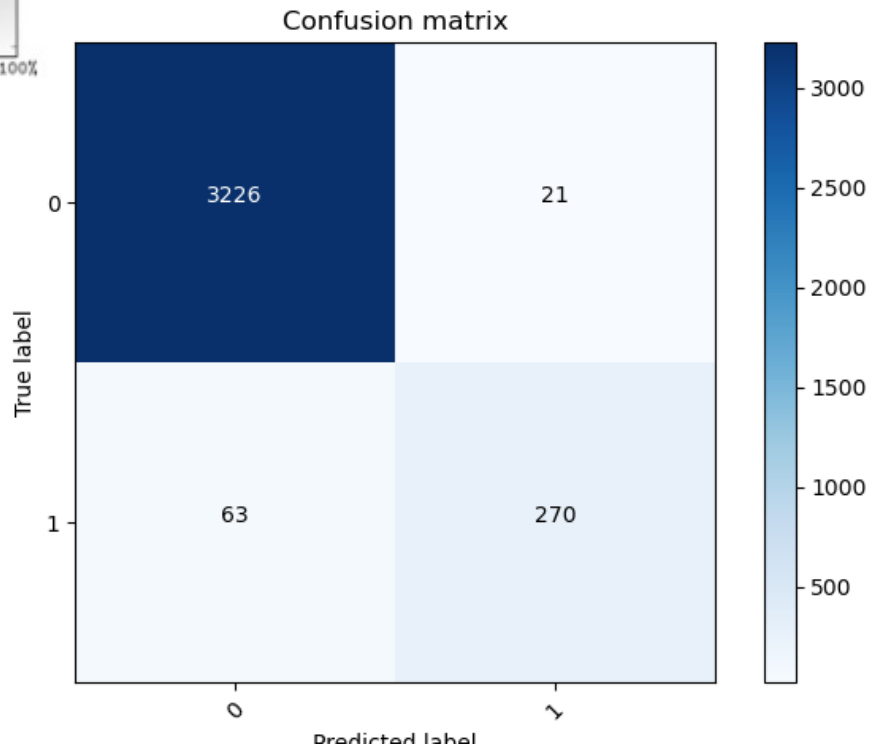
多项式核函数

Polynomial Kernel



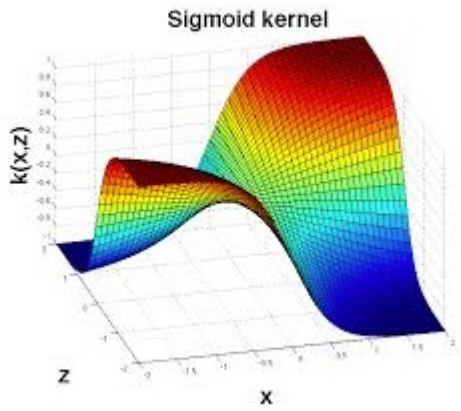
$$K(x_i , x_j) = (\gamma x_i^T x_j + r)^d , \gamma > 0$$

```
def svm_polynomial():
    print("##############  polynomial kernel ##############")
    kernel = 'poly'
    svm_train(kernel, C=1.0)

    svm_train(kernel, C=100.0)

    svm_train(kernel, C=1000.0)
```



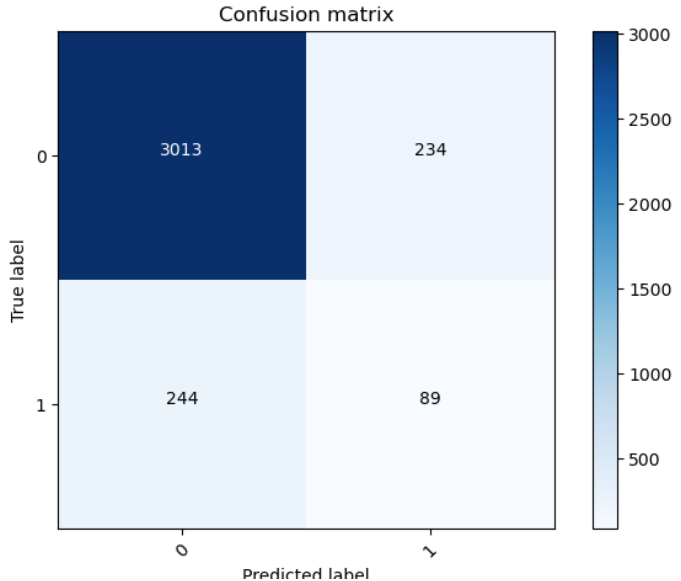
Sigmod核函数

sigmoid kernel : k (x, y) = tanh(αxTy + c)




```
kernel:sigmoid, C:10000.0, gamma:auto hyper parameters
Train Model accuracy score: 0.8744
Train Model accuracy score: 0.8668
Train Model cross-validation scores:
 [0.87604749 0.86941341 0.88438701 0.86866923 0.87041565]
Test Model cross-validation scores :
 [0.85614525 0.85893855 0.87011173 0.86731844 0.87011173]
Train Model Average stratified  cross-validation scores 0.8738
Test Model Average stratified cross-validation scores: 0.8645
True Positives(TP) =  3013
True Negatives(TN) =  90
False Positives(FP) =  234
False Negatives(FN) =  243
```

```
def svm_sigmoid():
    print("#############  polynomial kernel #############")
    kernel = 'sigmoid'
    svm_train(kernel, C=1.0)

    svm_train(kernel, C=100.0)

    svm_train(kernel, C=1000.0)
```



Grid Search查找最优算法

```
def grid_search():
    svc = SVC()
    # declare parameters for hyperparameter tuning
    parameters = [{'C': [1, 10, 100, 1000], 'kernel': ['linear']},
                  {'C': [1, 10, 100, 1000], 'kernel': ['rbf'], 'gamma':
                      [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]},
                  {'C': [1, 10, 100, 1000], 'kernel': ['poly'], 'degree': [2, 3, 4],
                   'gamma': [0.01, 0.02, 0.03, 0.04, 0.05]}
                  ]
    with parallel_backend('threading', n_jobs=8):
        grid_search = GridSearchCV(estimator=svc,
                                   param_grid=parameters,
```

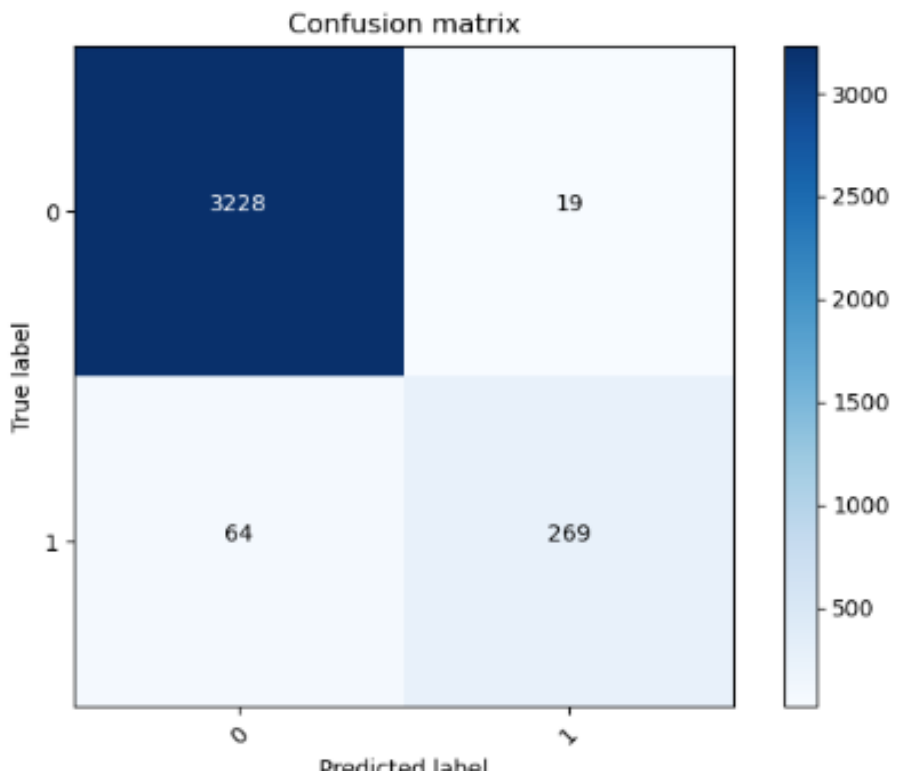
8线程

```
GridSearch CV best score : 0.9804
Parameters that give the best results : {'C': 10, 'kernel': 'linear'}
Estimator that was chosen by the search : SVC(C=10, kernel='linear')
GridSearch CV score on test set: 0.9768
```

```
    print('Parameters that give the best results :', (grid_search.best_params_))
    # print estimator that was chosen by the GridSearch
    print('Estimator that was chosen by the search :', (grid_search.best_estimator_))
    print('GridSearch CV score on test set: {0:0.4f}'.format(grid_search.score(svm.x_test, svm.y_test)))
```

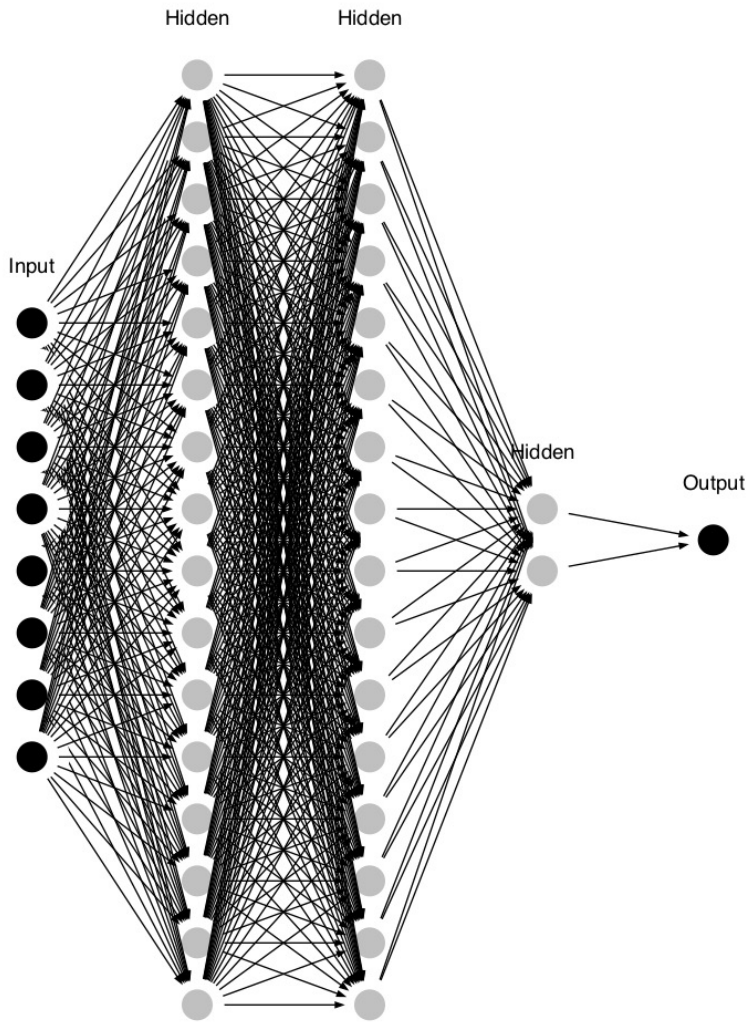
SVM最优结果分析

```
############## linear kernel #############
kernel:linear, C:10, gamma:auto hyper parameters
Train Model accuracy score: 0.9803
Train Model accuracy score: 0.9768
Train Model cross-validation scores:
 [0.97835196 0.9825419  0.98218652 0.97659797 0.98113867]
Test Model cross-validation scores :
 [0.98463687 0.97765363 0.97206704 0.97346369 0.97486034]
Train Model Average stratified  cross-validation scores 0.9802
Test Model Average stratified cross-validation scores: 0.9765
Confusion matrix [[3228   19]
 [  64  269]]
True Positives(TP) =  3228
True Negatives(TN) =  269
False Positives(FP) =  19
False Negatives(FN) =  64
Classification accuracy : 0.9768
Classification error : 0.0232
Precision : 0.9941
Recall or Sensitivity : 0.9806
False Positive Rate : 0.0660
Specificity : 0.9340
```



在SVM分类中使用线性核函数，C=10时可以达到最好的效果（原始数据中负样本率是90.7%）：
分类精度为97.68%，查准率是99.41%，召回率是98.06%，误判率是6.6%，特异度93.4%。

深度神经网络




```python
class NeuralNetwork(nn.Module):
    def __init__(self, num_classes):
        super(NeuralNetwork, self).__init__()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(8, 16),
            nn.ReLU(),
            nn.Linear(16, 16),
            nn.ReLU(),
            nn.Linear(16, num_classes),
            nn.Sigmoid()
        )

    def forward(self, x):
        outputs = self.linear_relu_stack(x)
        return outputs
```

模型参数

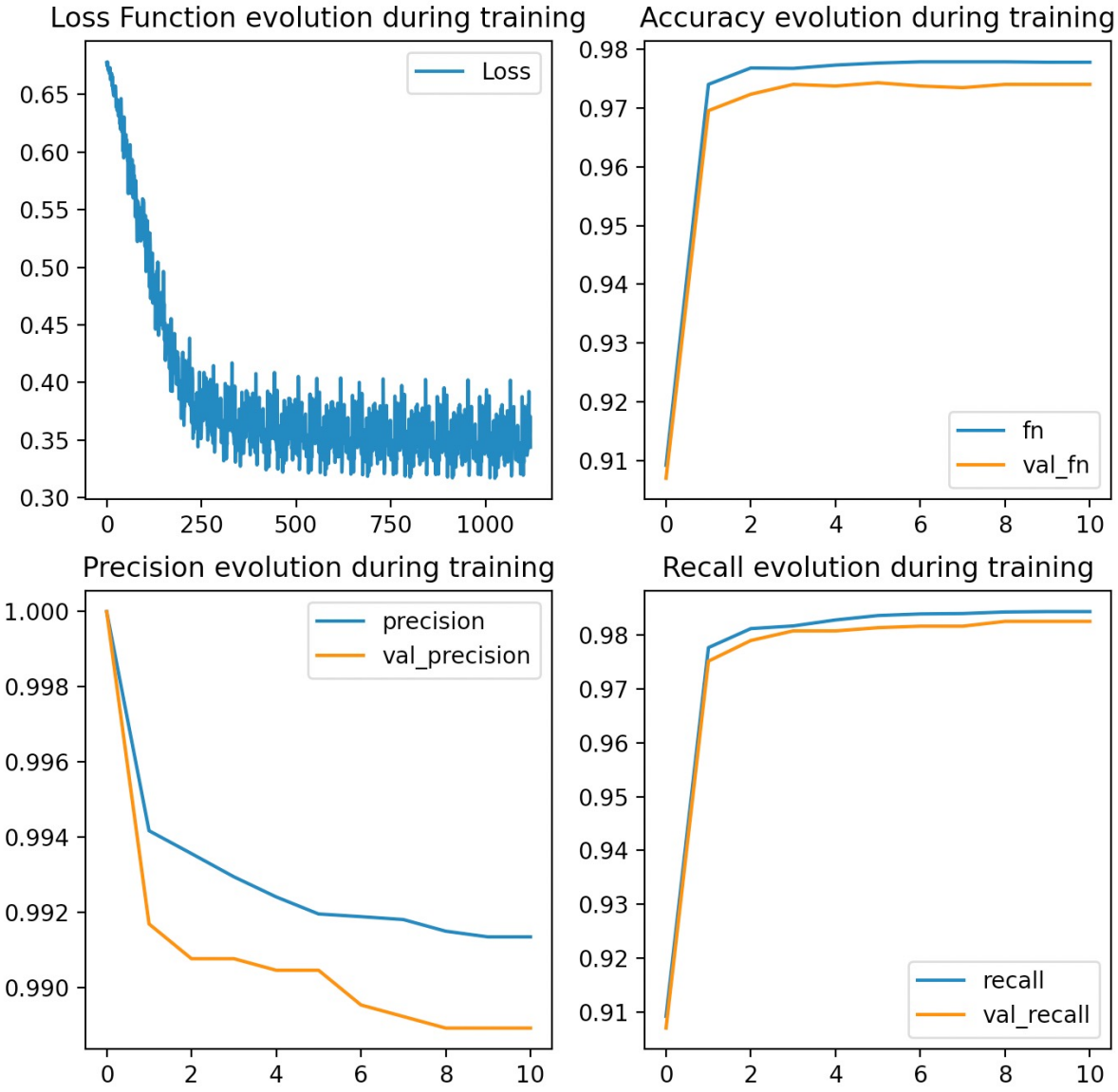
```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
NeuralNetwork                            [1, 2]                    --
├─Sequential: 1-1                        [1, 2]                    --
│    └─Linear: 2-1                       [1, 16]                   144
│    └─ReLU: 2-2                         [1, 16]                   --
│    └─Linear: 2-3                       [1, 16]                   272
│    └─ReLU: 2-4                         [1, 16]                   --
│    └─Linear: 2-5                       [1, 2]                    34
│    └─Sigmoid: 2-6                      [1, 2]                    --
==========================================================================================
Total params: 450
Trainable params: 450
Non-trainable params: 0
Total mult-adds (M): 0.00
==========================================================================================
```

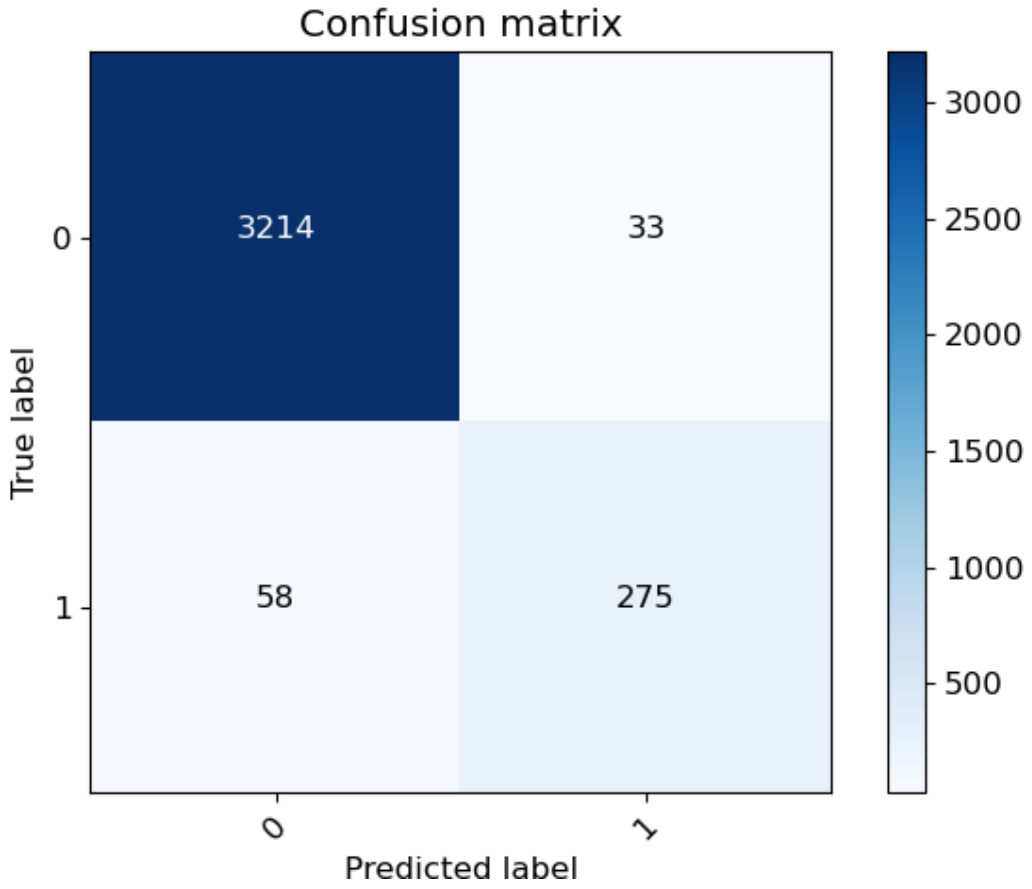
144=16*8+16

271=16*16+16

34=2*16+2

实验结果


Loss Function evolution during training
Loss
Accuracy evolution during training
fn
val_fn
Precision evolution during training
precision
val_precision
Recall evolution during training
recall
val_recall


Confusion matrix
3214
33
58
275
True label
Predicted label

残差网络

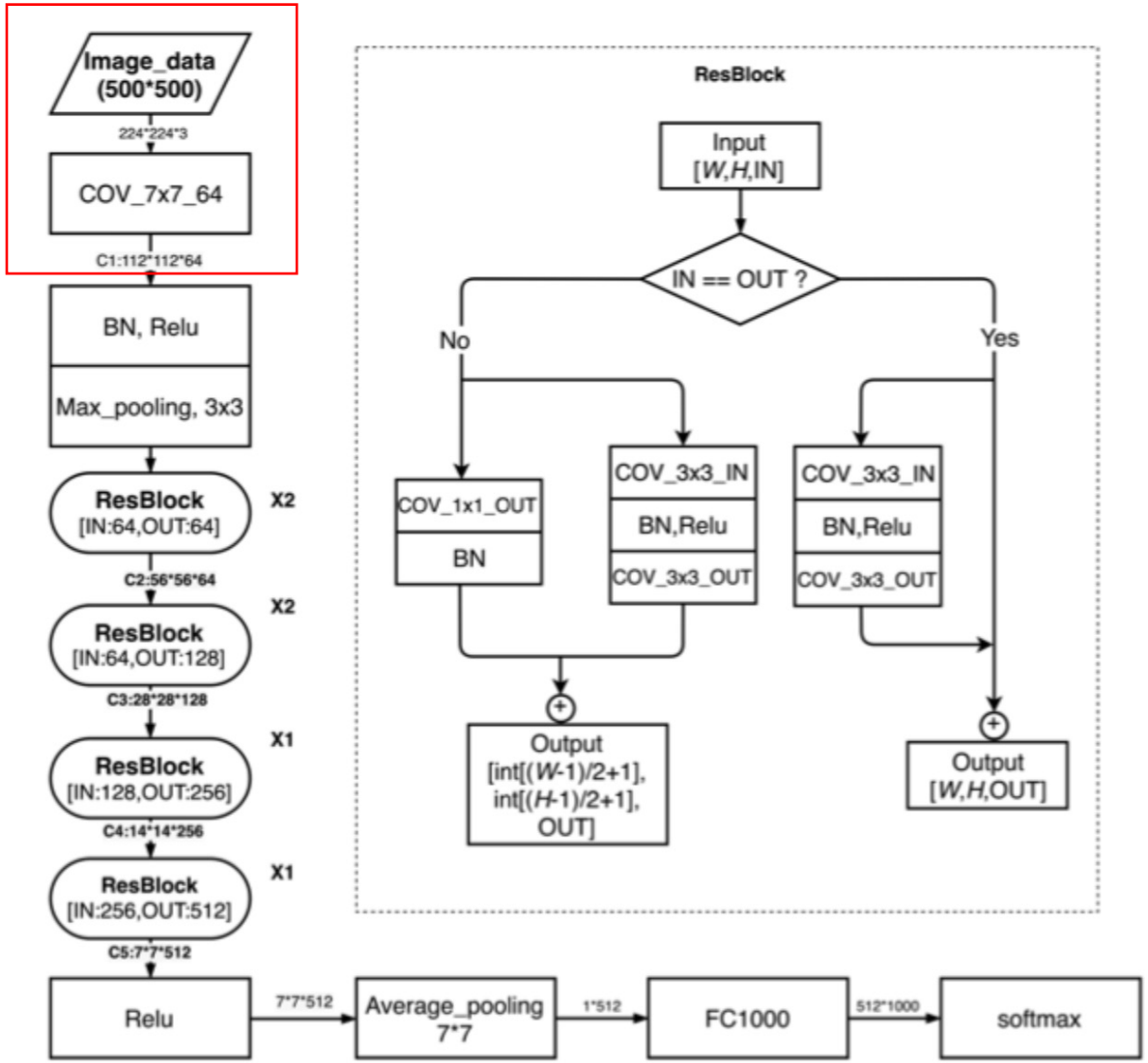


图 3　模型架构[30]

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
ResNet                                   [1, 2]                    --
├─Sequential: 1-1                        [1, 8, 1]                 --
│    └─ResidualBlock: 2-1                [1, 8, 1]                 --
│    │    └─Sequential: 3-1              [1, 8, 1]                 416
│    │    └─Sequential: 3-2              [1, 8, 1]                 --
│    └─ResidualBlock: 2-2                [1, 8, 1]                 --
│    │    └─Sequential: 3-3              [1, 8, 1]                 416
│    │    └─Sequential: 3-4              [1, 8, 1]                 --
├─Sequential: 1-2                        [1, 16, 1]                --
│    └─ResidualBlock: 2-3                [1, 16, 1]                --
│    │    └─Sequential: 3-5              [1, 16, 1]                1,216
│    │    └─Sequential: 3-6              [1, 16, 1]                160
│    └─ResidualBlock: 2-4                [1, 16, 1]                --
│    │    └─Sequential: 3-7              [1, 16, 1]                1,600
│    │    └─Sequential: 3-8              [1, 16, 1]                --
├─Sequential: 1-3                        [1, 16, 1]                --
│    └─ResidualBlock: 2-5                [1, 16, 1]                --
│    │    └─Sequential: 3-9              [1, 16, 1]                1,600
│    │    └─Sequential: 3-10             [1, 16, 1]                288
├─Sequential: 1-4                        [1, 16, 1]                --
│    └─ResidualBlock: 2-6                [1, 16, 1]                --
│    │    └─Sequential: 3-11             [1, 16, 1]                1,600
│    │    └─Sequential: 3-12             [1, 16, 1]                288
├─Linear: 1-5                            [1, 2]                    34
==========================================================================================
Total params: 7,618
Trainable params: 7,618
Non-trainable params: 0
Total mult-adds (M): 0.01
==========================================================================================
```

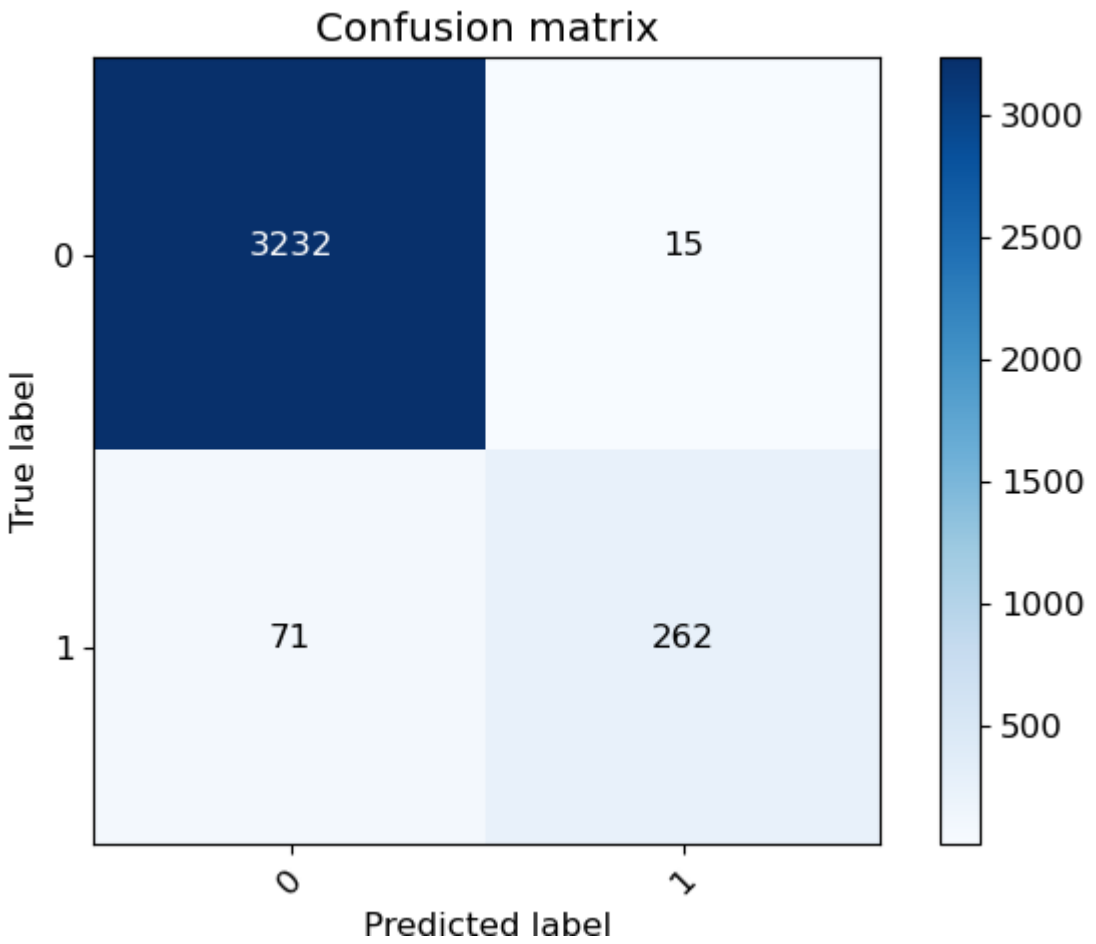
网络定义

```python
class ResidualBlock(nn.Module):
    def __init__(self, inchannel, outchannel, stride=1):
        super(ResidualBlock, self).__init__()
        self.left = nn.Sequential(
            nn.Conv1d(inchannel, outchannel, kernel_size=3,
                      stride=stride, padding=1, bias=False),
            nn.BatchNorm1d(outchannel),
            nn.ReLU(inplace=True),
            nn.Conv1d(outchannel, outchannel, kernel_size=3,
                      stride=1, padding=1, bias=False),
            nn.BatchNorm1d(outchannel)
        )
        self.shortcut = nn.Sequential()
        if stride != 1 or inchannel != outchannel:
            self.shortcut = nn.Sequential(
                nn.Conv1d(inchannel, outchannel, kernel_size=1,
                          stride=stride, bias=False),
                nn.BatchNorm1d(outchannel)
            )
    def forward(self, x):
        out = self.left(x)
        out = out + self.shortcut(x)
        out = F.relu(out)
        return out
```

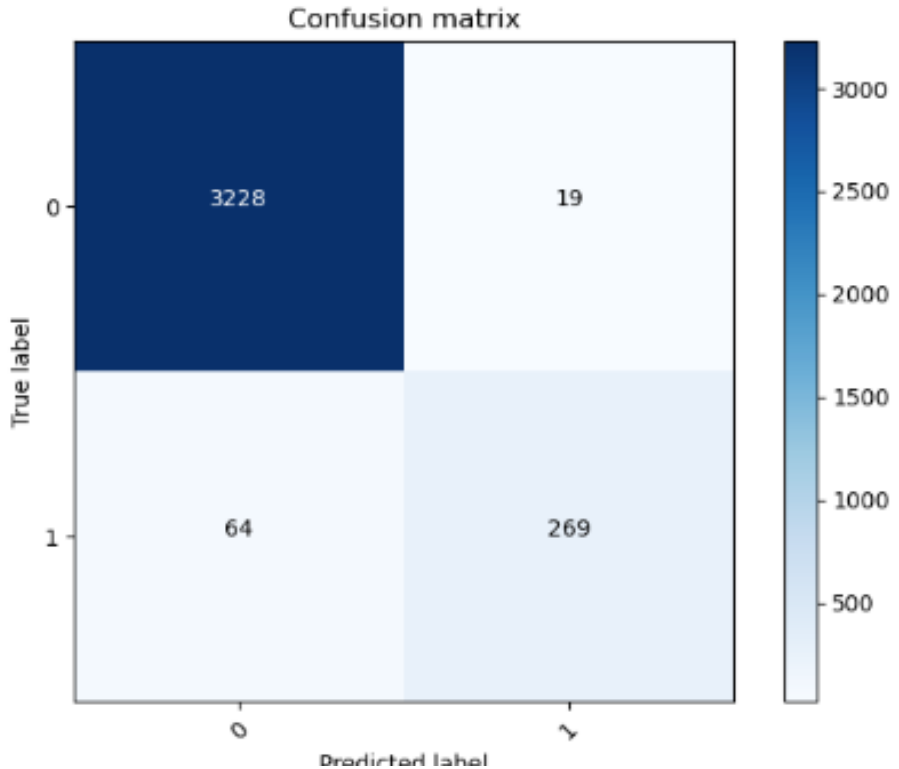
```python
class ResNet(nn.Module):
    def __init__(self, ResidualBlock, num_classes=10):
        super(ResNet, self).__init__()
        self.inchannel = 8
        self.layer1 = self.make_layer(ResidualBlock, 8, 2, stride=1)
        self.layer2 = self.make_layer(ResidualBlock, 16, 2, stride=2)
        self.layer3 = self.make_layer(ResidualBlock, 16, 1, stride=2)
        self.layer4 = self.make_layer(ResidualBlock, 16, 1, stride=2)
        self.fc = nn.Linear(16, num_classes)

    def make_layer(self, block, channels, num_blocks, stride):
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(self.inchannel, channels, stride))
            self.inchannel = channels
        return nn.Sequential(*layers)

    def forward(self, x):
        out = x[:, None, :]
        out = out.permute(0, 2, 1)
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool1d(out, 1)
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        return out
```

实验结果

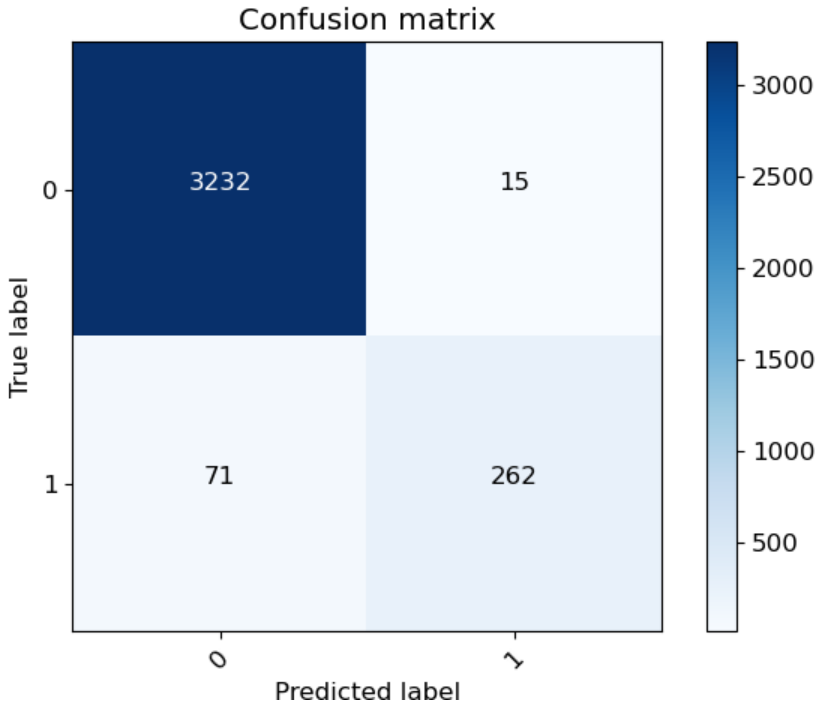
```
True Positives(TP) =  3232
True Negatives(TN) =  262
False Positives(FP) =  15
False Negatives(FN) =  71
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      3247
           1       0.95      0.79      0.86       333

    accuracy                           0.98      3580
   macro avg       0.96      0.89      0.92      3580
weighted avg       0.98      0.98      0.97      3580

Classification accuracy : 0.9760
Classification error : 0.0240
Precision : 0.9954
Recall or Sensitivity : 0.9785
False Positive Rate : 0.0542
Specificity : 0.9458
Test's ac is: 97.598%
```



对比结果



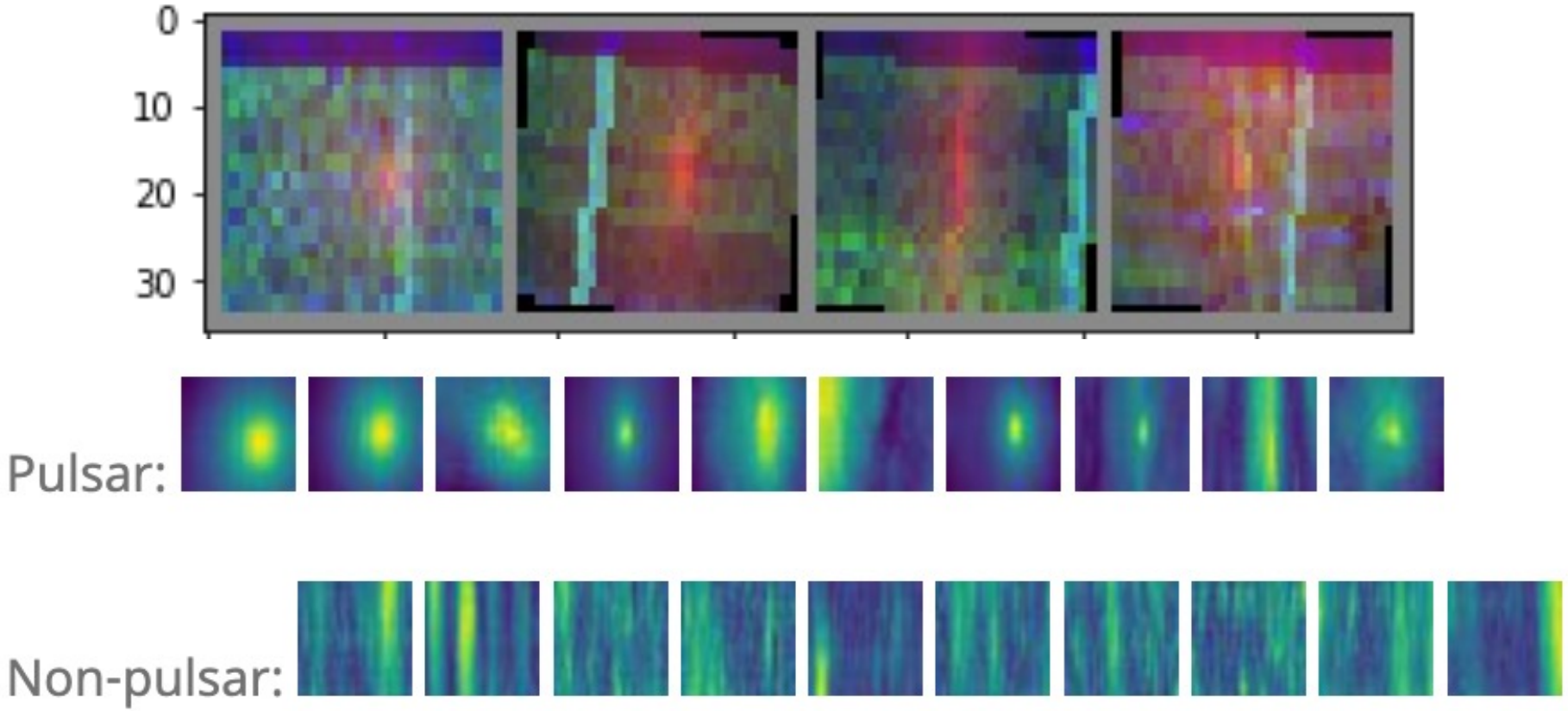
在SVM分类中使用线性核函数，C=10时可以达到最好的效果（原始数据中负样本率是90.7%）：
分类精度为97.68%，查准率是99.41%，召回率是98.06%，误判率是6.6%，特异度93.4%。



在CNN/ResNet分类中，最好的效果（原始数据中负样本率是90.7%）：
分类精度为97.60%，查准率是99.54%，召回率是97.85%，误判率是5.42%，特异度94.58%。

And More

如果直接处理图像数据呢？



图片数据集：https://as595.github.io/HTRU1/

处理图片的残差网络

```python
class ResidualBlock(nn.Module):
    def __init__(self, inchannel, outchannel, stride=1):
        super(ResidualBlock, self).__init__()
        # 这里定义了残差块内连续的2个卷积层
        self.left = nn.Sequential(
            nn.Conv2d(inchannel, outchannel, kernel_size=3, stride=stride, padding=1, bias=False),
            nn.BatchNorm2d(outchannel),
            nn.ReLU(inplace=True),
            nn.Conv2d(outchannel, outchannel, kernel_size=3, stride=1, padding=1, bias=False),
            nn.BatchNorm2d(outchannel)
        )
        self.shortcut = nn.Sequential()
        if stride != 1 or inchannel != outchannel:
            # shortcut，这里为了跟2个卷积层的结果结构一致，要做处理
            self.shortcut = nn.Sequential(
                nn.Conv2d(inchannel, outchannel, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(outchannel)
            )
    def forward(self, x):
        out = self.left(x)
        # 将2个卷积层的输出跟处理过的x相加，实现ResNet的基本结构
        out = out + self.shortcut(x)
        out = F.relu(out)
        return out
```

网络参数

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
ResNet                                   [1, 2]                    --
├─Sequential: 1-1                        [1, 64, 56, 56]           --
│    └─Conv2d: 2-1                       [1, 64, 56, 56]           1,728
│    └─BatchNorm2d: 2-2                  [1, 64, 56, 56]           128
│    └─ReLU: 2-3                         [1, 64, 56, 56]           --
├─Sequential: 1-2                        [1, 64, 56, 56]           --
│    └─ResidualBlock: 2-4                [1, 64, 56, 56]           --
│    │    └─Sequential: 3-1              [1, 64, 56, 56]           73,984
│    │    └─Sequential: 3-2              [1, 64, 56, 56]           --
│    └─ResidualBlock: 2-5                [1, 64, 56, 56]           --
│    │    └─Sequential: 3-3              [1, 64, 56, 56]           73,984
│    │    └─Sequential: 3-4              [1, 64, 56, 56]           --
├─Sequential: 1-3                        [1, 64, 28, 28]           --
│    └─ResidualBlock: 2-6                [1, 64, 28, 28]           --
│    │    └─Sequential: 3-5              [1, 64, 28, 28]           73,984
│    │    └─Sequential: 3-6              [1, 64, 28, 28]           4,224
├─Sequential: 1-4                              [1, 128, 14, 14]          --
│    └─ResidualBlock: 2-8                      [1, 128, 14, 14]          --
│    │    └─Sequential: 3-9                    [1, 128, 14, 14]          221,696
│    │    └─Sequential: 3-10                   [1, 128, 14, 14]          8,448
├─Sequential: 1-5                              [1, 256, 7, 7]            --
│    └─ResidualBlock: 2-9                      [1, 256, 7, 7]            --
│    │    └─Sequential: 3-11                   [1, 256, 7, 7]            885,760
│    │    └─Sequential: 3-12                   [1, 256, 7, 7]            33,280
├─Linear: 1-6                                  [1, 2]                    514
==========================================================================================
Total params: 1,451,714
Trainable params: 1,451,714
Non-trainable params: 0
Total mult-adds (M): 676.58
==========================================================================================
```

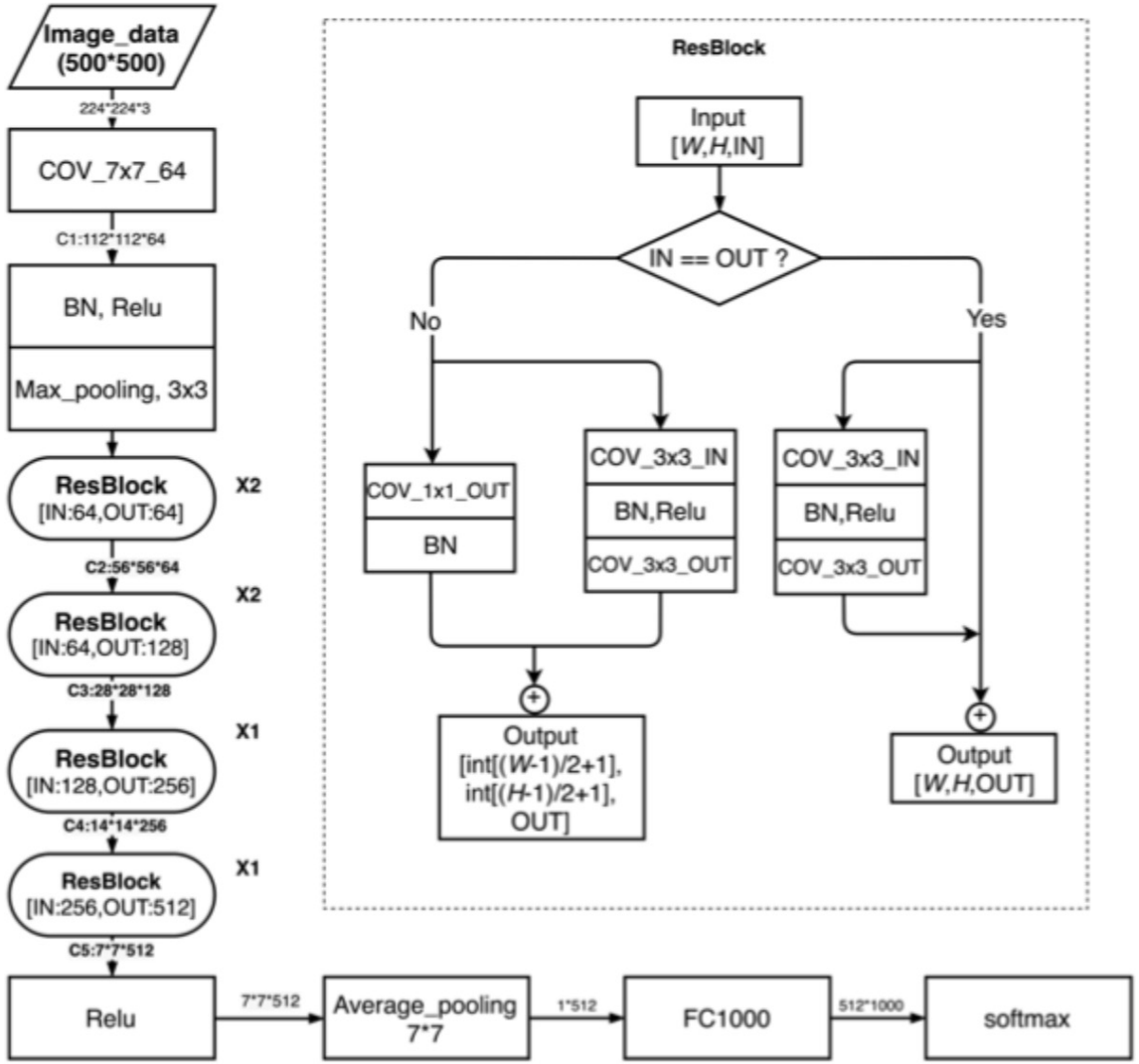
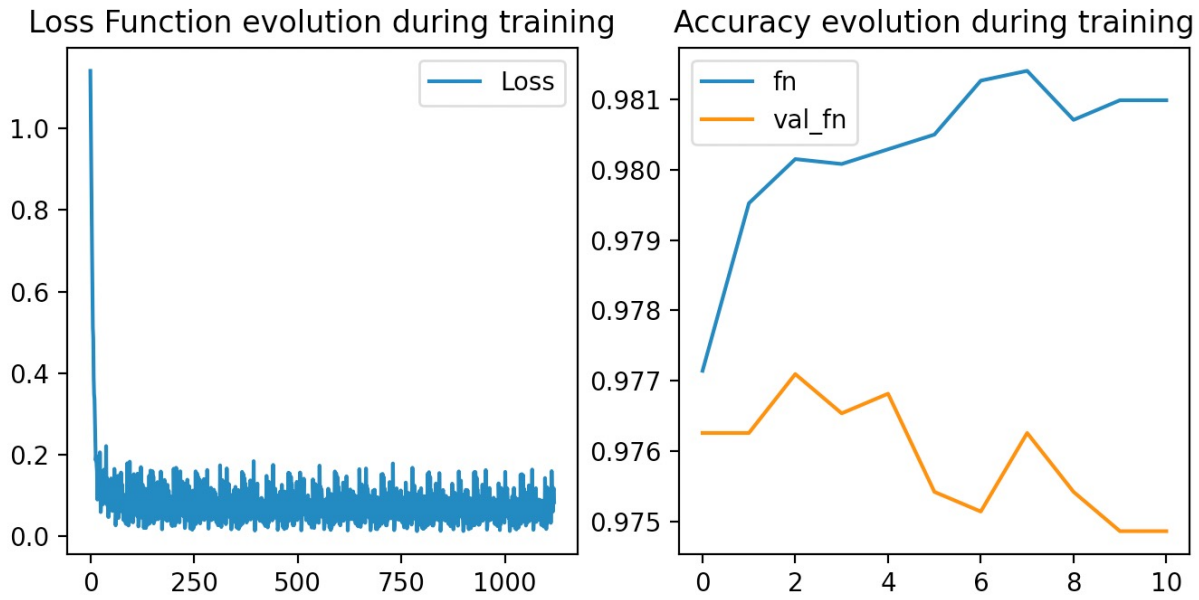
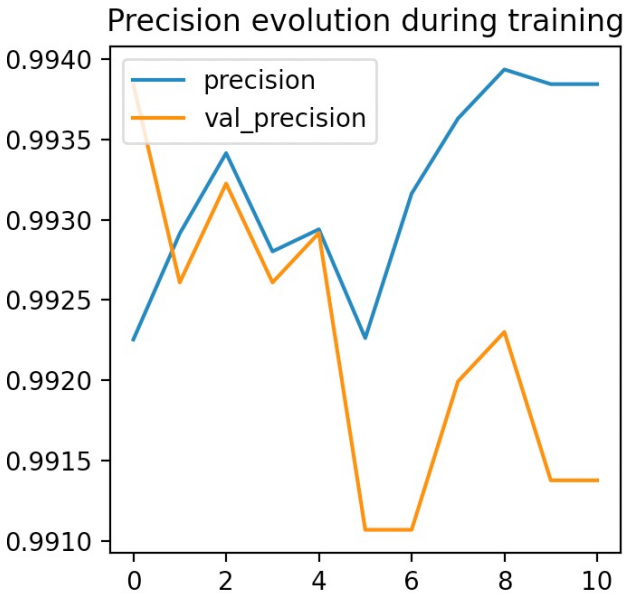
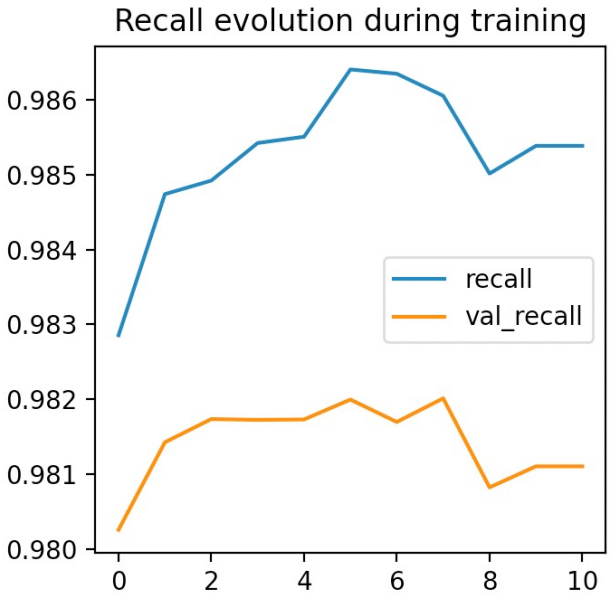


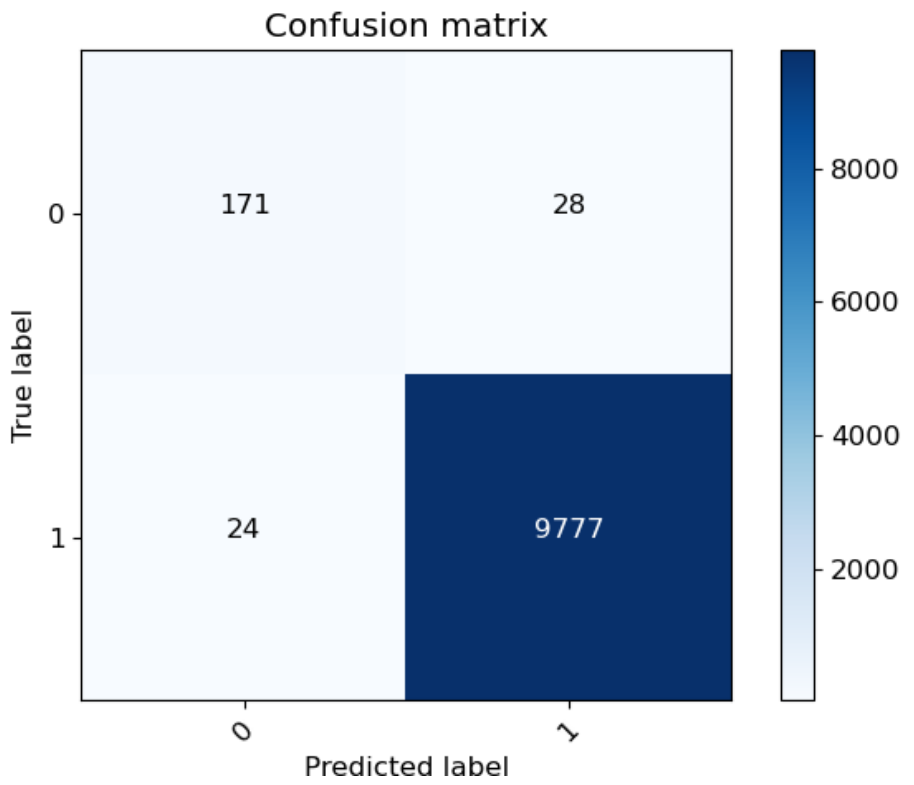
图 3　模型架构[30]

实验结果









在图片ResNet分类中，2轮训练后的效果（原始数据中负样本率是90.7%）：
分类精度为99.48%，查准率是85.93%，召回率是87.69%，误判率是0.29%，特异度99.71%。

总结

SVM分类办法：

优点：
1. 逻辑清晰，实现简单，内部逻辑解释性强
2. 训练、测试时间短，资源消耗少
3. 精确度相对比较高

缺点：
1. 针对不同的数据集需要多次调整比较不同参数的效果
2. 算法效果达到最好后难以继续提升
3. 无法很好地处理图片数据

深度神经网络分类办法：

优点：
1. 准确度高，可探索改进空间大
2. 算法复用及泛化能力强
3. 可通过卷积方式有效处理图片数据

缺点：
1. 逻辑模糊，实现复杂，内部逻辑解释性差
2. 训练、测试时间长，资源消耗多

参考链接

- SVM Classifier Tutorial
- 天眼科学目标：脉冲星的观测与研究意义
- 脉冲星候选样本分类方法综述：中国科学院 天文大数据中心
- 脉冲星数据比对分析和可视化系统设计与实现
- Undersampling and oversampling imbalanced data
- HTRU1 Pulsar Star
- HTRU2 Pulsar Star
- ResNet