Sankaranarayanan

06/21/2017

Approach

The given market data is extracted , class imbalance is negated and finally data is feature engineered. Keras open source neural network is used to build and artificial neural network on top of the engineered data. The hidded layers are altered, specific solvers are used keeping in mind the binary outcome. Rmsprop has been known to work with binary outcomes and hence it is used to build the ANN. Suitable loss functions and metrics are introduced to measure the goodness of the model. Optimal epoch size is selected to narrow down on the suitable ROC results. Precision Recall ROC and Area Under the Receiver Operating Characteristic curve are estimated and suitable plots are created to analyze optimal AUROC.

Estimated AUC on test.csv = 0.850748669861

Challenge in creation

Optimality of solvers, epoch size and hidden layers are critical in constructing this model. The feature selection is also challenging and RandomizedLogisticRegression proved to be more effective amongst other selection methods. The creation of Random forest ensemble models , decision trees with bagging and SVC were also tried, but teh Keras ANN was the one which exhibited maximum improvement in results at each model improvement stage

In [98]:
```
import os
import time
import keras
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import label_binarize
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import f1_score, roc_auc_score, precision_score, recall_s
core, accuracy_score, average_precision_score, precision_recall_curve, hamming
_loss
from sklearn.cross_validation import train_test_split
from sklearn.preprocessing import Imputer
from sklearn.model_selection import ShuffleSplit
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve,auc
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn import preprocessing
from pandas.tools.plotting import scatter_matrix
#intitalize neural network
from keras.models import Sequential
#create ANN layers
from keras.layers import Dense
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from sklearn import metrics
from sklearn import cross_validation
from sklearn.metrics import roc_auc_score
from sklearn.linear_model import RandomizedLogisticRegression
from sklearn.decomposition import PCA
```

In [99]:
```
def imputevals(dataset):
    """
    Operations:
    1. Column wise check for missing values
    2. Impute with most frequent if columns are of type Object
    3. Impute with mean if columns are not of type Object
    4. return dataset
    """

    k= pd.Series([dataset[c].value_counts().index[0]
        if dataset[c].dtype == np.dtype('O') else dataset[c].mean() for c in d
ataset],
        index=dataset.columns)
    return dataset.fillna(k)
```

In [100]:
```
def catencode(x):
    """
    Operations:
    1. Encode categorical variables
    2. Escape the dummy variable trap
    3. return encoded X
    """



    lblencoder_x = LabelEncoder()

    #dummy variable encoder
    #onehotencoder_x = OneHotEncoder(categorical_features = [0])

    x[:,2]=lblencoder_x.fit_transform(x[:,2])
    #onehotencoder = OneHotEncoder(categorical_features = [2])

    x[:,3]=lblencoder_x.fit_transform(x[:,3])
    #onehotencoder = OneHotEncoder(categorical_features = [3])
    #Xf = onehotencoder.fit_transform(X).toarray()
    onehotencoder = OneHotEncoder(categorical_features = [2,3])
    X = onehotencoder.fit_transform(x).toarray()
    X= np.delete(X,0,1)
    X= np.delete(X,4,1)


    return X
```

In [103]:
```
def scalerfunc(x):
    """
    Scales all indepedant variables so that computational
    efficiency is improved for our model
    """
    scaler = StandardScaler()
    x_scal = scaler.fit_transform(x)

    return x_scal
```

In [101]:
```
def ANN(classifier,X_train,y_train,X_test,y_test,n,Xtestdata):

    """
    ANN model
    Parameters used:
    out_dim - number of nodes in input layer
    init - Initialize weights to small numbers close to 0
    (Glorot with weights sampled from the normal distribution)
    activation - rectifier function , relu
    input_dim - number of input or indepedant features
    """
```

```
    classifier.add(Dense(output_dim = 6, init = 'uniform', activation =
'relu', input_dim = 8))

    # Adding the hidden layers
    classifier.add(Dense(output_dim = 6, init = 'glorot_normal', activation =
'relu'))
    classifier.add(Dense(output_dim = 6, init = 'glorot_normal', activation =
'relu'))
    classifier.add(Dense(output_dim = 5, init = 'glorot_normal', activation =
'relu'))
    classifier.add(Dense(output_dim = 5, init = 'glorot_normal', activation =
'relu'))
    classifier.add(Dense(output_dim = 5, init = 'glorot_normal', activation =
'relu'))

    # Adding the output layer
    classifier.add(Dense(output_dim = 1, init = 'glorot_normal', activation =
'sigmoid'))

    # Compiling the ANN
    classifier.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', me
trics = ['accuracy'])

    classifier.fit(X_train, y_train, batch_size = 10, nb_epoch = 100)

    # Part 3 - Making the predictions and evaluating the model

    # Predicting the Test set results
    #pred = classifier.predict_proba(X_test)
    y_pred = classifier.predict_proba(X_test)
    y_testpred = classifier.predict_proba(Xtestdata)

    """
    y_pred=[]
    for x in pred:
        if x>0.5:
            pred.append[1]
        else:
            pred.append[0]
    """

    y_predc = classifier.predict_proba(X_test)

    # Compute Precision-Recall and plot curve
    precision = dict()
    recall = dict()
    average_precision = dict()
    for i in range(n):
        precision[i], recall[i], _ = precision_recall_curve(y_test[:,i],
                                                            y_pred[:,i])
        average_precision[i] = average_precision_score(y_test[:,i],
y_pred[:,i])

    # Compute micro-average ROC curve and ROC area
    precision["micro"], recall["micro"], _ = precision_recall_curve(y_test.rav
el(),
        y_pred.ravel())
```

```
    average_precision["micro"] = average_precision_score(y_test, y_pred,
                                                         average="micro")
    # Visualizing the results


    # Plot Precision-Recall curve
    plt.clf()
    plt.plot(recall[0], precision[0], lw=2, color='navy',
             label='Precision-Recall curve')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title('Precision-Recall example: PR AUC={0:0.3f}'.format(average_preci
sion[0]))
    plt.legend(loc="lower left")
    plt.show()


    # Compute ROC curve and ROC area for each class
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n):
        fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_pred[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Compute micro-average ROC curve and ROC area
    fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_pred.ravel())
    roc_auc["micro"] = auc(fpr["micro"],tpr["micro"])

    plt.figure()
    lw = 2
    plt.plot(fpr[0], tpr[0], color='navy',
             lw=lw, label='ROC curve (area = %0.3f)' % roc_auc[0])
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()


    aucval = metrics.roc_auc_score(y_test, y_pred)



    return aucval,y_testpred
```

In [102]:
```
def readdata(directory,filename):
    os.chdir(directory)
    dataset = pd.read_csv(filename)
    return dataset
```

In [104]:
```
def imbalanced(dataset):
    zero = dataset[trainset['outcome'] == 0]
    one = dataset[trainset['outcome'] == 1]
    # get the data of each class
    if len(one) > len(zero):
        index = one.index[len(zero):]
    else:
        index = zero.index[len(one):]

    dataset.drop(index, inplace = True)
    return dataset
```

In [107]:
```
if __name__  == "__main__":
    print("Start of main")
    print("Import data")
    directory= os.getcwd()
    filename = 'train.csv'
    trainset = readdata(directory,filename)
    filename = 'test.csv'
    testset = readdata(directory,filename)

    trainset = imputevals(trainset)
    testset = imputevals(testset)
    print("Missing data has been imputed")

    print("Solve class Imbalance in trainingset")
    trainset = imbalanced(trainset)

    x= trainset.iloc[:,:-1].values

    y=trainset.iloc[:,9].values


    print("Encoding")

    X= catencode(x)
    xtestdata= testset.iloc[:,:].values

    Xtestdata= catencode(xtestdata)

    print("Categorical data has been encoded")

    print("Scaling data")
    X_scal= scalerfunc(X)
    Xtestdata= scalerfunc(Xtestdata)

    print("Feature Selection")

    rlr = RandomizedLogisticRegression()
    x_train = rlr.fit_transform(X_scal,y)
    pca = PCA(n_components=8)
    Xtestdata=pca.fit_transform(Xtestdata)
    print("Train Test Split")
    y = label_binarize(y, classes=[0, 1])
    n = y.shape[1]
    X_train, X_test, y_train, y_test = train_test_split(x_train, y, test_size
= 0.3, random_state = 1)

    classifier = Sequential()
    ROauc,testprediction=ANN(classifier,X_train,y_train,X_test,y_test,n,Xtestd
ata)
```

```
Start of main
Import data
Missing data has been imputed
Solve class Imbalance in trainingset
Encoding
Categorical data has been encoded
Scaling data
Feature Selection
Train Test Split
Epoch 1/100
1374/1374 [==============================] - 9s - loss: 0.6833 - acc: 0.5837

Epoch 2/100
1374/1374 [==============================] - 1s - loss: 0.6268 - acc: 0.7111

Epoch 3/100
1374/1374 [==============================] - 1s - loss: 0.5822 - acc: 0.7314

Epoch 4/100
1374/1374 [==============================] - 1s - loss: 0.5602 - acc: 0.7278

Epoch 5/100
1374/1374 [==============================] - 1s - loss: 0.5472 - acc: 0.7358

Epoch 6/100
1374/1374 [==============================] - 1s - loss: 0.5408 - acc: 0.7329

Epoch 7/100
1374/1374 [==============================] - 1s - loss: 0.5363 - acc: 0.7409

Epoch 8/100
1374/1374 [==============================] - 1s - loss: 0.5323 - acc: 0.7482

Epoch 9/100
1374/1374 [==============================] - 0s - loss: 0.5301 - acc: 0.7445

Epoch 10/100
1374/1374 [==============================] - 0s - loss: 0.5276 - acc: 0.7518

Epoch 11/100
1374/1374 [==============================] - 0s - loss: 0.5268 - acc: 0.7525

Epoch 12/100
1374/1374 [==============================] - 0s - loss: 0.5256 - acc: 0.7460

Epoch 13/100
1374/1374 [==============================] - 0s - loss: 0.5241 - acc: 0.7525

Epoch 14/100
1374/1374 [==============================] - 0s - loss: 0.5234 - acc: 0.7547

Epoch 15/100
1374/1374 [==============================] - 0s - loss: 0.5232 - acc: 0.7475

Epoch 16/100
1374/1374 [==============================] - 0s - loss: 0.5210 - acc: 0.7555
```

```
Epoch 17/100
1374/1374 [==============================] - 0s - loss: 0.5202 - acc: 0.7504

Epoch 18/100
1374/1374 [==============================] - 1s - loss: 0.5200 - acc: 0.7504

Epoch 19/100
1374/1374 [==============================] - 1s - loss: 0.5185 - acc: 0.7489

Epoch 20/100
1374/1374 [==============================] - 1s - loss: 0.5163 - acc: 0.7569

Epoch 21/100
1374/1374 [==============================] - 1s - loss: 0.5159 - acc: 0.7562

Epoch 22/100
1374/1374 [==============================] - 1s - loss: 0.5140 - acc: 0.7606

Epoch 23/100
1374/1374 [==============================] - 1s - loss: 0.5130 - acc: 0.7598

Epoch 24/100
1374/1374 [==============================] - 0s - loss: 0.5132 - acc: 0.7576

Epoch 25/100
1374/1374 [==============================] - 0s - loss: 0.5121 - acc: 0.7649

Epoch 26/100
1374/1374 [==============================] - 0s - loss: 0.5098 - acc: 0.7591

Epoch 27/100
1374/1374 [==============================] - 0s - loss: 0.5115 - acc: 0.7598

Epoch 28/100
1374/1374 [==============================] - 0s - loss: 0.5097 - acc: 0.7584

Epoch 29/100
1374/1374 [==============================] - 0s - loss: 0.5087 - acc: 0.7635

Epoch 30/100
1374/1374 [==============================] - 0s - loss: 0.5083 - acc: 0.7627

Epoch 31/100
1374/1374 [==============================] - 0s - loss: 0.5071 - acc: 0.7576

Epoch 32/100
1374/1374 [==============================] - 0s - loss: 0.5071 - acc: 0.7627

Epoch 33/100
1374/1374 [==============================] - 0s - loss: 0.5063 - acc: 0.7649

Epoch 34/100
1374/1374 [==============================] - 0s - loss: 0.5053 - acc: 0.7700

Epoch 35/100
1374/1374 [==============================] - 0s - loss: 0.5058 - acc: 0.7591
```

```
Epoch 36/100
1374/1374 [==============================] - 0s - loss: 0.5047 - acc: 0.7627

Epoch 37/100
1374/1374 [==============================] - 0s - loss: 0.5048 - acc: 0.7598

Epoch 38/100
1374/1374 [==============================] - 0s - loss: 0.5033 - acc: 0.7591

Epoch 39/100
1374/1374 [==============================] - 0s - loss: 0.5039 - acc: 0.7671

Epoch 40/100
1374/1374 [==============================] - 0s - loss: 0.5031 - acc: 0.7664

Epoch 41/100
1374/1374 [==============================] - 0s - loss: 0.5012 - acc: 0.7656

Epoch 42/100
1374/1374 [==============================] - 0s - loss: 0.5032 - acc: 0.7613

Epoch 43/100
1374/1374 [==============================] - 0s - loss: 0.5014 - acc: 0.7671

Epoch 44/100
1374/1374 [==============================] - 0s - loss: 0.5009 - acc: 0.7693

Epoch 45/100
1374/1374 [==============================] - 0s - loss: 0.5019 - acc: 0.7606

Epoch 46/100
1374/1374 [==============================] - 0s - loss: 0.5009 - acc: 0.7627

Epoch 47/100
1374/1374 [==============================] - 0s - loss: 0.4977 - acc: 0.7693

Epoch 48/100
1374/1374 [==============================] - 0s - loss: 0.5019 - acc: 0.7671

Epoch 49/100
1374/1374 [==============================] - 0s - loss: 0.5005 - acc: 0.7737

Epoch 50/100
1374/1374 [==============================] - 0s - loss: 0.5004 - acc: 0.7693

Epoch 51/100
1374/1374 [==============================] - 0s - loss: 0.4996 - acc: 0.7686

Epoch 52/100
1374/1374 [==============================] - 1s - loss: 0.5011 - acc: 0.7693

Epoch 53/100
1374/1374 [==============================] - 1s - loss: 0.4991 - acc: 0.7649

Epoch 54/100
1374/1374 [==============================] - 1s - loss: 0.4990 - acc: 0.7729
```

```
Epoch 55/100
1374/1374 [==============================] - 1s - loss: 0.4979 - acc: 0.7758

Epoch 56/100
1374/1374 [==============================] - 1s - loss: 0.4978 - acc: 0.7678

Epoch 57/100
1374/1374 [==============================] - 1s - loss: 0.4995 - acc: 0.7627

Epoch 58/100
1374/1374 [==============================] - 0s - loss: 0.4968 - acc: 0.7744

Epoch 59/100
1374/1374 [==============================] - 0s - loss: 0.4982 - acc: 0.7656

Epoch 60/100
1374/1374 [==============================] - 0s - loss: 0.4985 - acc: 0.7649

Epoch 61/100
1374/1374 [==============================] - 0s - loss: 0.4962 - acc: 0.7678

Epoch 62/100
1374/1374 [==============================] - 0s - loss: 0.4982 - acc: 0.7678

Epoch 63/100
1374/1374 [==============================] - 0s - loss: 0.4973 - acc: 0.7664

Epoch 64/100
1374/1374 [==============================] - 0s - loss: 0.4951 - acc: 0.7656

Epoch 65/100
1374/1374 [==============================] - 0s - loss: 0.4968 - acc: 0.7744

Epoch 66/100
1374/1374 [==============================] - 0s - loss: 0.4962 - acc: 0.7744

Epoch 67/100
1374/1374 [==============================] - 0s - loss: 0.4971 - acc: 0.7715

Epoch 68/100
1374/1374 [==============================] - 0s - loss: 0.4953 - acc: 0.7686

Epoch 69/100
1374/1374 [==============================] - 0s - loss: 0.4976 - acc: 0.7649

Epoch 70/100
1374/1374 [==============================] - 0s - loss: 0.4977 - acc: 0.7700

Epoch 71/100
1374/1374 [==============================] - 0s - loss: 0.4960 - acc: 0.7642

Epoch 72/100
1374/1374 [==============================] - 0s - loss: 0.4947 - acc: 0.7715

Epoch 73/100
1374/1374 [==============================] - 0s - loss: 0.4952 - acc: 0.7693
```

```
Epoch 74/100
1374/1374 [==============================] - 0s - loss: 0.4961 - acc: 0.7744

Epoch 75/100
1374/1374 [==============================] - 0s - loss: 0.4965 - acc: 0.7649

Epoch 76/100
1374/1374 [==============================] - 0s - loss: 0.4965 - acc: 0.7686

Epoch 77/100
1374/1374 [==============================] - 0s - loss: 0.4961 - acc: 0.7671

Epoch 78/100
1374/1374 [==============================] - 0s - loss: 0.4964 - acc: 0.7693

Epoch 79/100
1374/1374 [==============================] - 0s - loss: 0.4944 - acc: 0.7751

Epoch 80/100
1374/1374 [==============================] - 0s - loss: 0.4953 - acc: 0.7686

Epoch 81/100
1374/1374 [==============================] - 0s - loss: 0.4952 - acc: 0.7686

Epoch 82/100
1374/1374 [==============================] - 0s - loss: 0.4958 - acc: 0.7700

Epoch 83/100
1374/1374 [==============================] - 0s - loss: 0.4939 - acc: 0.7737

Epoch 84/100
1374/1374 [==============================] - 0s - loss: 0.4936 - acc: 0.7737

Epoch 85/100
1374/1374 [==============================] - 0s - loss: 0.4938 - acc: 0.7686

Epoch 86/100
1374/1374 [==============================] - 0s - loss: 0.4943 - acc: 0.7722

Epoch 87/100
1374/1374 [==============================] - 0s - loss: 0.4947 - acc: 0.7737

Epoch 88/100
1374/1374 [==============================] - 0s - loss: 0.4941 - acc: 0.7664

Epoch 89/100
1374/1374 [==============================] - 0s - loss: 0.4938 - acc: 0.7700

Epoch 90/100
1374/1374 [==============================] - 0s - loss: 0.4926 - acc: 0.7671

Epoch 91/100
1374/1374 [==============================] - 0s - loss: 0.4934 - acc: 0.7642

Epoch 92/100
1374/1374 [==============================] - 0s - loss: 0.4925 - acc: 0.7787
```

```
Epoch 93/100
1374/1374 [==============================] - 0s - loss: 0.4924 - acc: 0.7686

Epoch 94/100
1374/1374 [==============================] - 0s - loss: 0.4942 - acc: 0.7700

Epoch 95/100
1374/1374 [==============================] - 0s - loss: 0.4926 - acc: 0.7715

Epoch 96/100
1374/1374 [==============================] - 0s - loss: 0.4945 - acc: 0.7744

Epoch 97/100
1374/1374 [==============================] - 0s - loss: 0.4927 - acc: 0.7722

Epoch 98/100
1374/1374 [==============================] - 0s - loss: 0.4928 - acc: 0.7678

Epoch 99/100
1374/1374 [==============================] - 0s - loss: 0.4893 - acc: 0.7780

Epoch 100/100
1374/1374 [==============================] - 0s - loss: 0.4921 - acc: 0.7693

352/590 [================>.............] - ETA: 0s
```

Results

AUC

In [81]:
```
print(ROauc)
```

```
0.850748669861
```

The above results where got when the randomized logistic regression method of feture selection selected 8 features for modelling, it is also possible that the sleection method may select 7 features. In that case the selection process is re run as we have the maximum results with the 8 most prominent features selected

In [96]:
```
print("Predicted probability values of positive outcome for test.csv" )
testprediction
```

```
Predicted probability values of positive outcome for test.csv
```

Out[96]:
```
array([[ 0.31915501],
       [ 0.44646215],
       [ 0.93447131],
       ...,
       [ 0.34713653],
       [ 0.52658176],
       [ 0.72054708]], dtype=float32)
```

In [94]:
```
print( "Setting a hard probability limit of 0.5 to segregate between the two c
lasses")

test_pred=[]
for x in testprediction:
    if x>0.5:
            test_pred.append(1)
    else:
            test_pred.append(0)
```

```
Setting a hard probability limit of 0.5 to segregate between the two classes
```

In [97]:
```
print ("The outcome for test.csv")


test_pred
```

The outcome for test.csv

```
Out[97]: [0,
          0,
          1,
          0,
          0,
          1,
          0,
          0,
          0,
          0,
          1,
          1,
          0,
          0,
          1,
          1,
          1,
          1,
          0,
          1,
          0,
          1,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          1,
          0,
          1,
          0,
          0,
          1,
          0,
          0,
          1,
          1,
          1,
          0,
          1,
          0,
          1,
          0,
          0,
          0,
          1,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
          0,
```

```
0,
0,
0,
0,
1,
0,
0,
1,
0,
1,
0,
1,
0,
1,
0,
1,
1,
0,
1,
1,
0,
1,
1,
0,
0,
1,
1,
1,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,
1,
0,
0,
0,
1,
1,
0,
1,
0,
0,
0,
1,
0,
0,
0,
0,
0,
1,
0,
0,
0,
```

```
1,
0,
1,
0,
1,
1,
0,
0,
0,
0,
0,
0,
0,
0,
1,
0,
1,
1,
1,
1,
1,
1,
0,
0,
1,
0,
0,
0,
0,
0,
0,
1,
0,
1,
0,
0,
1,
1,
0,
0,
0,
0,
0,
0,
0,
1,
0,
0,
1,
0,
0,
0,
1,
1,
0,
0,
1,
1,
```

```
0,
1,
1,
0,
1,
1,
1,
1,
0,
1,
0,
1,
0,
0,
1,
0,
1,
1,
0,
0,
0,
1,
1,
0,
1,
0,
0,
0,
0,
1,
1,
1,
0,
0,
0,
0,
0,
0,
0,
0,
1,
0,
1,
0,
0,
1,
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
1,
```

```
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
0,
1,
0,
0,
0,
0,
0,
0,
0,
1,
0,
0,
0,
0,
0,
1,
0,
0,
0,
1,
1,
0,
1,
0,
1,
0,
0,
0,
0,
1,
1,
0,
0,
1,
1,
0,
0,
1,
1,
0,
0,
1,
1,
1,
0,
```

```
0,
0,
0,
1,
0,
0,
1,
1,
1,
0,
1,
0,
1,
1,
0,
0,
0,
1,
1,
1,
0,
0,
0,
0,
1,
1,
1,
1,
0,
1,
0,
1,
0,
1,
0,
1,
1,
1,
1,
0,
0,
1,
1,
1,
1,
0,
1,
0,
0,
0,
0,
0,
0,
1,
0,
0,
```

```
1,
0,
1,
0,
1,
0,
0,
0,
0,
1,
0,
0,
1,
1,
0,
1,
0,
0,
1,
0,
0,
1,
1,
0,
0,
0,
1,
0,
0,
0,
1,
1,
1,
0,
1,
0,
0,
0,
0,
1,
0,
0,
1,
1,
1,
1,
0,
0,
0,
1,
0,
0,
0,
0,
0,
1,
1,
1,
```

```
0,
0,
0,
1,
1,
0,
1,
1,
1,
0,
1,
0,
0,
0,
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,
1,
1,
1,
0,
1,
0,
1,
1,
0,
1,
0,
0,
0,
1,
1,
1,
1,
1,
0,
1,
1,
1,
0,
0,
1,
0,
1,
1,
1,
```

```
0,
0,
0,
0,
0,
0,
0,
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
0,
0,
0,
0,
1,
1,
0,
1,
0,
1,
0,
1,
0,
1,
1,
0,
1,
0,
1,
0,
0,
0,
1,
1,
0,
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
1,
0,
0,
0,
1,
1,
0,
0,
0,
```

```
0,
0,
0,
1,
0,
0,
1,
0,
0,
0,
0,
0,
1,
1,
1,
0,
0,
0,
1,
0,
1,
0,
1,
1,
0,
0,
0,
0,
0,
1,
0,
0,
0,
1,
0,
1,
0,
0,
1,
0,
1,
0,
1,
0,
1,
0,
1,
1,
0,
0,
0,
0,
1,
0,
0,
1,
0,
```

```
0,
0,
1,
1,
0,
0,
0,
0,
0,
0,
0,
1,
1,
0,
1,
1,
0,
0,
1,
0,
0,
0,
0,
1,
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
1,
0,
1,
0,
0,
1,
0,
0,
1,
0,
1,
1,
0,
0,
0,
0,
0,
0,
1,
0,
0,
0,
0,
0,
0,
```

```
0,
0,
0,
0,
0,
1,
0,
1,
0,
0,
0,
1,
0,
1,
0,
1,
0,
1,
1,
0,
0,
1,
0,
0,
0,
0,
1,
1,
0,
0,
0,
0,
1,
0,
0,
1,
0,
0,
0,
1,
0,
0,
0,
0,
0,
0,
0,
1,
1,
0,
0,
0,
1,
0,
1,
0,
```

```
0,
0,
1,
1,
0,
0,
0,
1,
0,
0,
1,
0,
0,
0,
1,
1,
1,
0,
0,
0,
0,
0,
0,
1,
0,
1,
0,
0,
0,
1,
0,
1,
1,
0,
1,
0,
0,
0,
1,
0,
0,
0,
0,
0,
0,
1,
0,
0,
1,
0,
1,
0,
1,
0,
0,
1,
1,
```

```
0,
0,
0,
1,
1,
0,
1,
0,
0,
1,
0,
0,
0,
0,
1,
1,
1,
1,
1,
0,
0,
1,
1,
0,
0,
0,
1,
0,
1,
0,
1,
1,
0,
0,
0,
0,
0,
0,
0,
0,
1,
0,
0,
0,
1,
0,
1,
0,
0,
1,
0,
0,
0,
1,
0,
0,
0,
```

```
1,
1,
1,
0,
1,
0,
0,
0,
0,
1,
0,
0,
1,
1,
0,
0,
0,
0,
0,
1,
0,
1,
1,
0,
0,
1,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
0,
1,
1,
0,
0,
0,
1,
0,
0,
1,
0,
0,
0,
0,
1,
0,
1,
0,
1,
1,
0,
0,
```

```
1,
1,
1,
1,
1,
0,
1,
0,
1,
0,
0,
0,
1,
0,
0,
0,
1,
0,
0,
0,
1,
1,
0,
0,
0,
1,
0,
0,
0,
1,
1,
0,
0,
1,
0,
0,
1,
0,
0,
1,
0,
0,
1,
0,
0,
0,
0,
0,
1,
1,
0,
0,
0,
0,
0,
0,
1,
1,
1,
```

```
0,
0,
1,
0,
1,
0,
0,
0,
0,
1,
1,
1,
1,
0,
1,
0,
0,
1,
1,
0,
0,
1,
0,
1,
1,
0,
1,
0,
0,
0,
0,
0,
1,
1,
1,
0,
0,
0,
0,
0,
1,
0,
1,
1,
1,
1,
1,
0,
0,
0,
1,
0,
0,
1,
1,
1,
0,
0,
0,
```