PROJECT 3: CREDIT CARD FRAUD DETECTION

ALI NEHRANI

Abstract. This project is about detecting credit-card Fraud from data provided from kaggle website. Feature V1 ... V28, "Time" and "Amount" can be used as data and "class" will used as target. I used 5-fold cross validation to measure the performance of the dataset and checked the accuracy with ROC and Area Under the Precision-Recall Curve (AUPRC).

1. Introduction

Binary classifiers are mostly statistical models that divide a dataset into two groups, Yes (1 usually for class of interest) and No (0 usualy for other class) [1]. They have been successfully applied to a wide range of biological and medical and also security problems in recent years [1, 2]. The performance of the classifier's prediction is very important deciding how informative the classifier could be, or how comparible with leading well known methods. Among the measures of classifier performance in the level of model construction are accuracy we can mention to error rate, and the Area under the Receiver Operating Characteristics (ROC) curve (AUC) which are more common.several plots provide visual representations, such as ROC and Precision-Recall (PRC) plots could be useful in evaluating of the classification model.

Class imbalancea difference in the numbers of positive and negative instances. ROC is the most popular evaluation method for binary classifiers, but the interpretation of ROC curves needs a nore attentions spesifically when we are dealing with imbalanced datasets.

2. Basic Concepts

If binary classifier become perfect, predicted labels will be exactly the same as data target. Figure 1 schematically represents the observed labels and perfect predicted labels which are fit together. As far as in practice we rarely have perfect predictor, therefore most if the times the results of predictor will have some degree of overlapping with the observed labels as it is shown in Figure 2. This overlapping contains some errors which in general is categurized into four parts:

- True positive (TP): correct positive prediction
- False positive (FP): incorrect positive prediction
- True negative (TN): correct negative prediction
- False negative (FN): incorrect negative prediction

The matrix representation of these errors leads to a confusion matrix concept which is very useful in describing accuracy measures.

Date: Dec. 2017.

Thanks for your time to read my report.

FIGURE 1. The actual class of observed data (left), results of perfect predictor (right) - from [3]

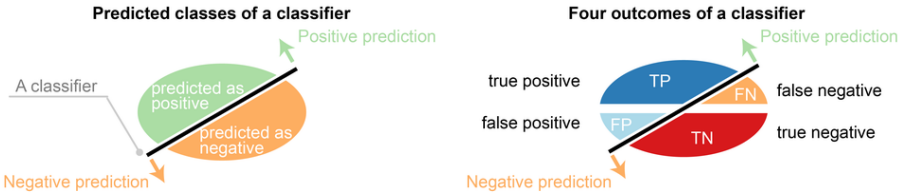

FIGURE 2. Overlapping of common classifiers with the observed labels (left), possible errors that raise up during classification [3]

2.1. **Confusion Matrix.** The confusion matrix is a two by two table that contains four outcomes produced by a binary classifier. Various measures including advanced measures, are derived from the confusion matrix. The entries of confusion matrix are as follows:

- $(1,1)$: TPs: number of correct positive prediction
- $(2,1)$: FPs: number of incorrect positive prediction
- $(1,2)$: FNs: number of incorrect negative prediction
- $(2,2)$: TN: number of correct negative prediction

Based onconfusion matrix the following error measures will appear:
1. Error rate (ERR) is ratio of incorrect predictions wrt total number of the dataset [3]:

$$Err = \frac{FP + FN}{P + N}$$

2. Accuracy (ACC) ratio of correct predictions wrt total number of the dataset:

$$ACC = \frac{TP + TN}{P + N}$$

3. ensitivity (SN) is number of correct positive predictions divided by the total number of positives:

$$SN = \frac{TP}{P}$$

4. Specificity (SP) is ratio of correct negative predictions wrt total number of negatives:

$$SP = \frac{TN}{N}$$

5. Positive predictive value (PREC) is ratio of correct positive predictions wrt total number of positive predictions

$$PREC = \frac{TP}{TP + FP}$$

6. False positive rate (FPR) is ratio of incorrect positive predictions wrt the total number of negatives:

$$FPR = \frac{FP}{TN + FP} = 1 - SP$$

and finaly F-score or harmonic mean of precision and recall:

$$\frac{(1 + \beta^2)PREC * REC}{\beta^2 PREC + REC}$$

2.2. **ROC curve interpretation.** As I use ROC curves in this project it would be quite useful if I describe in short the curve interpretation. The accuracy of the test mainly depends on how well the test separates the positive group into those with and without the positive label in the obsrvation.

The ROC curve of TPF (sensitivity) versus FPF (1-specificity) across varying

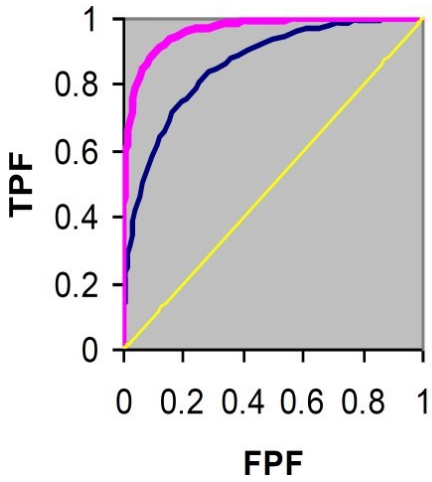

FIGURE 3. Interpreting ROC curve; Yellow- Random Classifier results, Blue- classifier with medium results and Pink- classifire with better results

cut-offs leads to a curve in the unit square. The slope of an ROC curve at any point is equal to the ratio of the two density functions describing, respectively, the distribution of the separator variable in the positive and negetive groups, i.e. the likelihood ratio [4]. A monotonically increasing likelihood ratio corresponds to a concave ROC curve [6].

The area under the curve (AUC) which is computed by numerical approximation of the area, describes the accuracy of the classifier [5]. The AUC is an effective and combined measure of sensitivity and specificity that describes the inherent validity of diagnostic tests [7]. In Figure 3 I represent the accuracy interpretation of the ROC curve.

In order to improve the performance of the classifier we should try to make the ROC curve to be close to the left and top axes, see Figure 4

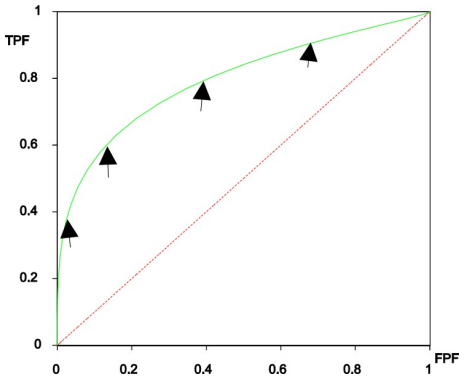

FIGURE 4. If the ROC curve become close to the left and top axis, it will be more accurate - the perfect one overlaps on left (y)- top (x) axes

3. Preparing Data

The data is downloaded from kaggle website, after stroing in local disc, it is loaded by the following command:

```
fileAddress=os.path.join(os.getcwd(), 'creditcard.csv')
print('Start to loading first data from this file: \n', fileAddress)
CredData = pd.read_csv(fileAddress)
```

The data is red by pandas dataframe and saved to work with the name "CredData". The collumns on this data is summerized as

```
colNames = CredData.columns.values
print(colNames)
print(CredData.describe())

['Time' 'V1' 'V2' 'V3' 'V4' 'V5' 'V6' 'V7' 'V8' 'V9' 'V10' 'V11' 'V12'
 'V13' 'V14' 'V15' 'V16' 'V17' 'V18' 'V19' 'V20' 'V21' 'V22' 'V23' '
    V24'
 'V25' 'V26' 'V27' 'V28' 'Amount' 'Class']
```

The class column consists of "1" for existing fraud and "0" for normal transaction. In the following I want to look at the distribution of Fraud data.

DISTRIBUTION OF THE DATA

In order to obtain the distribution of the fraud data we can simply count number of the Fraud and normal transaction. To this aim I creat two classes; class "0" for normal transaction and class "1" for Fraud transaction:

```
print("Normal transaction (class = 0):", CredData['Class'][CredData['
    Class'] == 0].count())
print("Fraudulent transaction (class = 1):", CredData['Class'][
    CredData['Class'] == 1].count())
```

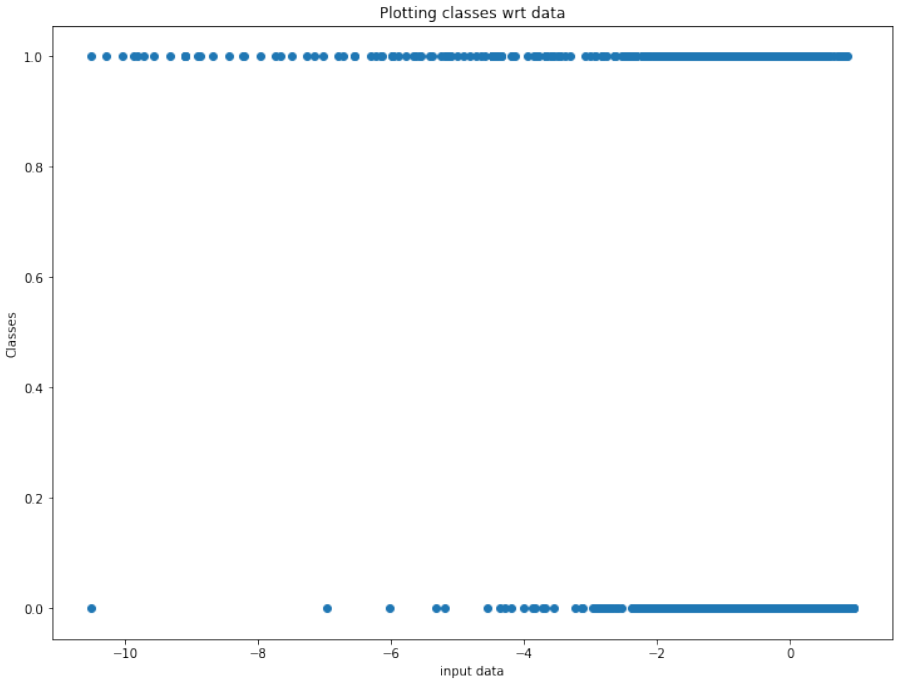

FIGURE 5. Feature V2 of the Data is plotted with respect to the classes; This Figure represents the structure of the V2 feature of the data distributed in two classes.

```
Normal transaction (class = 0): 284315
Fraudulent transaction (class = 1): 492
```

I separated the two classes for further data preparation using the sklearn DataFrame query over the column "class":

```
# Distribution of the fraud
class0 = CredData.query('Class == 0')
class1 = CredData.query('Class == 1')
```

Data.query works equivallent to

$$Data.query('Data.class == 0') = Data[Data[class] == 0].$$

And sampling from the data for shuffling

```
# shuffling the dataset
class0 = class0.sample(frac=1)
class1 = class1.sample(frac=1)
```

4. Creat Train and Test Data set

In this project I implemented two different method for dividing data into train and test. At the first one I use previously defined classes:

```
# data balancing
Train_0 = class0.iloc[0:6000]
Train_1 = class1
```

```
4
5 # combine
6 train = Train_0.append(Train_1, ignore_index=True).values
7
8 # split data into input and target
9 x_train = train[:,0:30].astype(float)
10 y_train = train[:,30]
11
12 # standardization
13 std_x_tr = preprocessing.StandardScaler().fit(x_train)
14 std_x_tr = std_x_tr.transform(x_train)
```

In this code, data is balanced (by location), the I combined training of the classes and finally I separated training input and target data and finally standardized them to obtain better results.

5. Data Classification

For training data, I applied LogisticRegression method from sklearn's linear-classification methods. In order to use other methods it is easy to replace this method with others. I the following LogisticGegression method:

```
1 # testing with logistic regression  - short solution
2 model = linear_model.LogisticRegression()
3 model = model.fit(std_x_tr,y_train)
4
5 # checking the accuracy of the model
6 print('Logistic Regression Score',model.score(std_x_tr,y_train))
```

In the following, predicted targets are reported and are used to measure "ROC" accuracy score, confusion matrix and sklearn classification report:

```
1 y_pred = model.predict(std_x_tr)
2 metrics.roc_auc_score(y_train, y_pred)
3 metrics.confusion_matrix(y_train, y_pred)
4 metrics.classification_report(y_train, y_pred)
```

Recall: `roc_auc_score` computes Area Under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores.
`confusion_matrix` is applied to evaluate the accuracy of a classification. By definition a confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in class 0 but predicted to be in class 1. Thus in binary classification, the count of true negatives is $C_{0,0}$, false negatives is $C_{1,0}$, true positives is $C_{1,1}$ and false positives is $C_{0,1}$.

```
1 # Probability estimation of the classes
2 y_pre_pro = model.predict_proba(std_x_tr)
3 metrics.roc_auc_score(y_train, y_pred_pro)
4 metrics.confusion_matrix(y_train, y_pred_pro)
5 metrics.classification_report(y_train, y_pred_pro)
```

`predict_proba` Probability estimates and returns estimates for all classes are ordered by the label of classes.
The results are reported in the following:

```
Logistic Regression Score 0.987985212569

ROC AUC Score 0.928195121951

Confusion Matrix:
                    [[5992    8]
                    [  70  422]]

Classification report:
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99      6000
         1.0       0.98      0.86      0.92       492
avg / total       0.99      0.99      0.99      6492
```

Computing Cross Validation Score. For more comparison and completion, I compute cross validation classification with $cv = 5$ folds and select ROC AUC as scoring method.

```
#scoring with area under pr curve with linea regression
scoring = 'roc_auc'
kfold=5
results = cross_val_score(model, x_train, y_train, cv=kfold, scoring=
    scoring)
print('The results of all 5 fold hand separated data:\n',results)
print("ROC AUC: mean = {}, std = {}\n".format(results.mean(), results.
    std()))
```

Results of all folds and also mean and std of results are reported in the following:

```
The results of all 5 fold hand separated data:
 [ 0.91494108  0.93698653  0.89685374  0.93056122  0.94477891]
AUC: mean = 0.9248242973957259, std = 0.017082440945791668
```

5.1. **ROC AU Curve.** A ROC AUC curve is a most commonly used way to visualize the performance of a binary classifier. ROC AUC indicates that the probability of a randomly selected positive example will be scored higher by the classifier than a randomly selected negative example. In the case of multiple models with nearly the same accuracy, we can pick the one that gives a higher ROC AUC. In the following code, ROC AUC score is computed and ploted:

```
# Determine the false positive and true positive rates
false_pos_rate, true_pos_rate, _ = metrics.roc_curve(y_train,
    y_pre_pro[:,0])

# Determine ROC AUC
roc_auc = metrics.auc(false_pos_rate, true_pos_rate)
print('ROC AUC', roc_auc)

# Plot of a ROC AUC curve for a specific class
plt.figure()
plt.plot( true_pos_rate,false_pos_rate, label='ROC AU curve (area =
    {})'.format(roc_auc))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
```

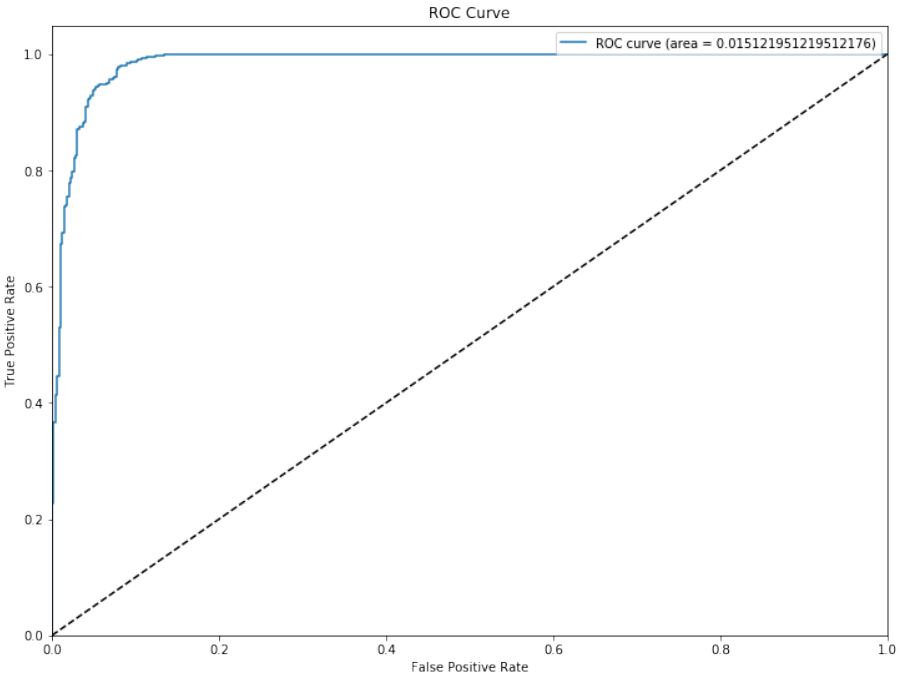

FIGURE 6. ROC AU Curve for LogisticRegression results; reported area is the area of upper part of the curve. The stright dash-line (- -) represents random classifier results

```
plt.ylim([0.0, 1.05])
plt.xlabel('True Positive Rate')
plt.ylabel('False Positive Rate')
plt.title('ROC AU Curve')
plt.legend(loc="best")
plt.show()
```

6. Classification with sklearn split data

The third approach for classification that I used in this report is designing train and test set with sklearn `train_test_split` method. I set `test_size`=0.3 that means 70% of data will be used for training. The classifier is set to be DecisionTreeClassifier:

```
# Splitting the data - using cross validation
x_train, x_test, y_train, y_test = train_test_split(x_train,y_train,
    test_size=.3, random_state = 0)
dt_clf = tree.DecisionTreeClassifier()
dt_clf.fit(x_train, y_train)
y_pr = dt_clf.predict(x_train)
```

Cross validation score is computed for training and test data:

```
# train score report
```

```
cv_train_score = cross_val_score(dt_clf, x_train, y_train, scoring='
    accuracy', cv=5)
cv_test_score = cross_val_score(dt_clf, x_test, y_test, scoring='
    accuracy', cv=5)

print('training cv score: {},\n testing cv score: {}'.format(
    cv_train_score, cv_test_score) )
print('training cv score mean:{} and std:{},\n testing cv score mean
    :{} and std:{}'.format(cv_train_score.mean(),cv_train_score.std(),
     cv_test_score.mean(),  cv_test_score.std()))
```

The following results are achieved:

```
training cv score: [ 0.97472527  0.98461538  0.97687225  0.97797357
    0.97687225],
 testing cv score: [ 0.98205128  0.96666667  0.96923077  0.97429306
    0.98457584]
training cv score mean:0.9782117442029337 and std
    :0.0033706769308831552,
 testing cv score mean:0.9753635225100521 and std:0.006985480571270245
```

6.1. **Stratified K-Fold Cross-Validation.** At the forth and final part of the project, I applied Stratified K-Fold Cross-Validation method for designing train and test data. Number of folds is set to 5 and again DecisionTreeClassifier is applied. Although linear SVC works better, but in order to apply different approches, I selected this classifier. For measuring the accuracy, first I reported the methods score:

```
# design the sampling
#I will use the previous model - Decision Tree
#
kfold = cross_validation.StratifiedKFold(y=y_train, n_folds=5,
    random_state=0)
#
train_score = []
test_score = []
for k, (itrain,itest) in enumerate(kfold):
    dt_clf.fit(x_train[itrain], y_train[itrain])
    train_score.append(dt_clf.score(x_train[itrain], y_train[itrain]))
    # score for test
    test_score.append(dt_clf.score(x_train[itest], y_train[itest]))
    print('Fold: {}, Train Accuracy: {}, Test  Accuracy: {}'.format(k
    +1, train_score, test_score))
print('\n Train CV accuracy: {}'.format(np.mean(train_score)))
print('Test CV accuracy: {}'.format(np.mean(test_score)))
```

In the following results of all folds and also sverage scores are obtained:

```
Fold: 1, Train Accuracy: [1.0], Test  Accuracy: [0.97032967032967032]
Fold: 2, Train Accuracy: [1.0, 1.0], Test  Accuracy:
    [0.97032967032967032, 0.9747252747252747]
Fold: 3, Train Accuracy: [1.0, 1.0, 1.0], Test  Accuracy:
    [0.97032967032967032, 0.9747252747252747, 0.98017621145374445]
Fold: 4, Train Accuracy: [1.0, 1.0, 1.0, 1.0], Test  Accuracy:
    [0.97032967032967032, 0.9747252747252747, 0.98017621145374445,
    0.97797356828193838]
```

```
Fold: 5, Train Accuracy: [1.0, 1.0, 1.0, 1.0, 1.0], Test  Accuracy:
    [0.97032967032967032, 0.9747252747252747, 0.98017621145374445,
    0.97797356828193838, 0.96365638766519823]
Train CV accuracy: 1.0
Test CV accuracy: 0.9733722224911652
```

7. Plotting Results

A ROC curve is created by connecting all ROC points of a classifier in the ROC space. Two adjacent ROC points can be connected by a straight line, and the curve starts at $(0.0, 0.0)$ and ends at $(1.0, 1.0)$.
It was shown that there is one-to-one relationship between area under ROC curve and precision, according to this the area under ROC curve is returned as measure of accuracy (See Figure 7).
In the final part of the project I presented the results by ROC AU curve. The plotting is done in the following code; Like previous section I designed train and test dataset by `StratifiedKFold` method. Number of folds is set to 5 and ROC AU curve is plotted for all folds and also the curve of the average results.
For plotting ROC AU curve I used `roc_curve` method (previously I plotted ROC curve by my self function).

```
# change size of Matplotlib plot
fig_size = plt.rcParams["figure.figsize"] # Get current size

old_fig_params = fig_size
# new figure parameters
fig_size[0] = 12
fig_size[1] = 9

plt.rcParams["figure.figsize"] = fig_size # set new size


kfold = StratifiedKFold(n_splits=5, random_state=7)
# plot roc-curve
# code adapted from http://scikit-learn.org
mean_tpr = 0.0
mean_fpr = np.linspace(0, 1, 100)

colors = cycle(['cyan', 'indigo', 'seagreen', 'magnet', 'blue', '
    darkorange'])
lw = 2

i = 0
for (train, test), color in zip(kfold.split(x_train, y_train), colors)
    :
    probas_ = model.fit(x_train[train], y_train[train]).predict_proba(
    x_train[test])
    # Compute ROC curve and area the curve
    fpr, tpr, thresholds = roc_curve(y_train[test], probas_[:, 1])
    mean_tpr += interp(mean_fpr, fpr, tpr)
    mean_tpr[0] = 0.0
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, lw=lw, color=color,
```

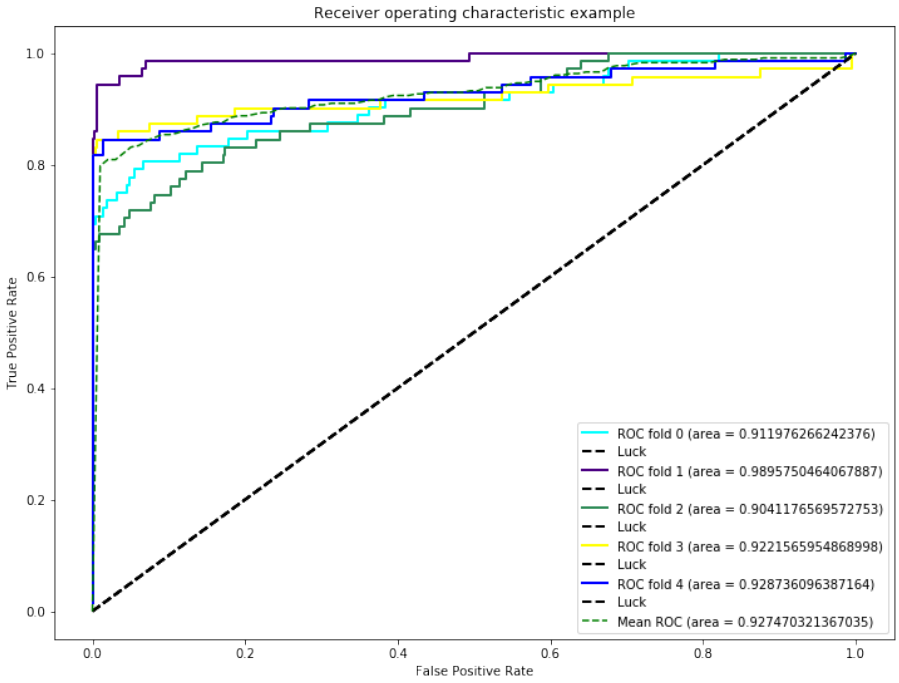

FIGURE 7. ROC AU curve for all 5 folds of train and test process. The curve of average results are also included. The stright dash-line (- -) represents random classifier results

```
            label='ROC fold {} (area = {})'.format(i, roc_auc))

    i += 1

    plt.plot([0, 1], [0, 1], linestyle='--', lw=lw, color='k',
             label='Good chance')

mean_tpr /= kfold.get_n_splits(x_train, y_train)
mean_tpr[-1] = 1.0
mean_auc = auc(mean_fpr, mean_tpr)
plt.plot(mean_fpr, mean_tpr, color='g', linestyle='--',
             label='Mean ROC (area = {})'.format(mean_auc, lw=lw))

plt.xlim([-0.05, 1.05])
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('The results of ROC analysis')
plt.legend(loc="best")
plt.show()
```

References

[1] Saito, Takaya AND Rehmsmeier, Marc, The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets, PLOS ONE, Public Library of Science, 10(2), 2015
[2] Berrar D, Flach P. Caveats and pitfalls of ROC analysis in clinical microarray research (and how to avoid them). Brief Bioinform. Oxford University Press; 2011 Mar 21;13(1):bbr00897.
[3] https://classeval.wordpress.com/introduction/basic-evaluation-measures/
[4] Greiner M, Pfeiffer D, Smith RD. Priciples and practical application of the receiver operating characteristic analysis for diagnostic test. Prev Vet Med. 2000;45:23-41.
[5] Swets JA. ROC analysis applied to the evaluation of medical imaging techniques. Invest Radiol. 1979;14:109-21.
[6] Swets JA. Indices of discrimination or diagnostic accuracy: their ROCs and implied models. Psychol Bull. 1986;99:100-17.
[7] Kummar R, Indrayan A. Receiver operating characteristic (ROC) curve for medical researchers. Indian Pediatr. 2011;48:27789.

Department of Computer Science, Ozyegin University
E-mail address: ali.nehrani@ozu.edu.tr