AUROC-vs-AUPRC

Diabetes -- Sheryl

Sheryl's data set is diabetes.csv, and the competition is from https://www.kaggle.com/uciml/pima-indians-diabetes-database/kernels?sortBy=voteCount&group=everyone&pageSize=20&datasetId=228&outputType=all&turbolinks%5BrestorationIdentifier%5D=26144c10-5644-4852-b3c4-50264072b98a

A useful guide for choosing ML evaluation metrics can be found in the file how-to-choose-right-metric-for-evaluating-ml-model.ipynb.

The main codes can be found from the file diabetes_Sheryl_20200117.ipynb.

Random Guess Classifier Analysis

The diabetes dataset has a total of 768 samples, 268 of which have positive labels, accounting for a percentage of around 35%. The dimension of the inputs is 8.

Now we do the random guess classifier analysis. If we are using a random guess classifier, then we will get AUROC = 0.5, AUPRC = 0.35, accuracy less than 0.65.

Batch Comparison

In fact, average precision is NOT the same as AUPRC, and the accuracy calculated after selecting the best threshold under AUROC may NOT be as good as the accuracy given by the prediction of the baseline models. The codes for this is in diabetes_Sheryl_1024.ipynb, which is a completely new file written by Sheryl from scratch. The most important part of the file that gives the comparison table is the following:

```
from sklearn.metrics import roc_auc_score, average_precision_score, f1_score,
log_loss, recall_score, precision_recall_curve, auc
from sklearn.metrics import roc_curve, accuracy_score
seed=7
models = [] # Here I will append all the algorithms that I will use. Each one
will run in all the created datasets.
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('RF', RandomForestClassifier()))
models.append(('AdaBoost', AdaBoostClassifier()))

# compare different classifiers
results_accuracy=[]
results_auroc=[]
results_average_precision=[]
results_neg_log_loss=[]
results_f1 = []
results_recall =[]
names=[]
```

```
fpr_full = []
tpr_full = []
thresholds_roc_full = []
precision_full = []
recall_full = []
thresholds_prc_full = []
measures =
['AUROC','AUPRC','accuracy_best_threshold','accuracy','average_precision','f1','
log_loss_score','recall']
scores_table = np.zeros([8,8])
roc_cut = np.zeros([8,]).astype(int) # cut points for fpr, tpr, thresholds for
ROC curve of each model
prc_cut = np.zeros([8,]).astype(int) # cut points for precision, recall,
thresholds for PRC curve of each model
i = 0 # looping index
for name, model in models:
        y_pred_proba = model.fit(X_train, y_train).predict_proba(X_test)[:, 1]
        fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
        fpr_full = np.concatenate((fpr_full, fpr))
        tpr_full = np.concatenate((tpr_full, tpr))
        thresholds_roc_full = np.concatenate((thresholds_roc_full, thresholds))
        roc_cut[i + 1] = roc_cut[i] + fpr.shape[0]
        #Area under ROC curve
        auroc = roc_auc_score(y_test,y_pred_proba)
        precision, recall, thresholds =
precision_recall_curve(y_test,y_pred_proba)
        precision_full = np.concatenate((precision_full, precision))
        recall_full = np.concatenate((recall_full, recall))
        thresholds_prc_full = np.concatenate((thresholds_prc_full, thresholds))
        prc_cut[i + 1] = prc_cut[i] + recall.shape[0]
        # area under PRC curve
        auprc = auc(recall, precision)

        threshold = Find_Optimal_Cutoff(y_test,y_pred_proba)
        y_pred = y_pred_proba>threshold
        accuracy_best_threshold = accuracy_score(y_test, y_pred)
        accuracy = accuracy_score(y_test, model.predict(X_test))
        average_precision = average_precision_score(y_test,
model.predict(X_test))
        f1 = f1_score(y_test, model.predict(X_test))
        log_loss_score = log_loss(y_test, model.predict(X_test))
        recall = recall_score(y_test, model.predict(X_test))
        names.append(name)


        # report of scores
        scores_table[i, 0] = auroc
        scores_table[i, 1] = auprc
        scores_table[i, 2] = accuracy_best_threshold
        scores_table[i, 3] = accuracy
        scores_table[i, 4] = average_precision
        scores_table[i, 5] = f1
        scores_table[i, 6] = -log_loss_score
        scores_table[i, 7] = recall
        print(name,': AUROC = {:.3f}, AUPRC = {:.3f}, average precision =
{:.3f}, '.format(auroc,auprc,average_precision),
              '. \nBest threshold for ROC = {:.3f},'.format(threshold[0]),
'accuracy for the best ROC threshold is then {:.3f},'
```

```
              .format(accuracy_best_threshold),'accuracy =
{:.3f}.'.format(accuracy),
              '\nF1 score = {:.3f},'.format(f1), 'log loss =
{:.3f},'.format(log_loss_score),'recall = {:.3f}.'.format(recall))
        print ("--"*30)
        i = i + 1


scores_table[7, 0] = 0.5 # random guess
scores_table[7, 1] = ratio # random guess
#plot ROC
plt.plot([0,1],[0,1],'k--')
for i in range(7):
    plt.plot(fpr_full[roc_cut[i]:roc_cut[i + 1]],tpr_full[roc_cut[i]:roc_cut[i +
1]], label=name)
plt.xlabel('fpr')
plt.ylabel('tpr')
title_name = 'diabetes ROC curve'
plt.title(title_name)
plt.legend(['random guess','LR', 'LDA','KNN','CART','NB','RF','AdaBoost'])
save_name = 'diabetes ROC curve.png'
plt.savefig(save_name)
plt.show()

# plot PRC
plt.axhline(y=ratio, xmin=0, xmax=1,color='k', linestyle = '--')
for i in range(7):
    plt.plot(recall_full[prc_cut[i]:prc_cut[i +
1]],precision_full[prc_cut[i]:prc_cut[i + 1]], label=name)
plt.xlabel('recall')
plt.ylabel('precision')
title_name = 'diabetes PRC curve'
plt.title(title_name)
plt.legend(['random guess','LR', 'LDA','KNN','CART','NB','RF','AdaBoost'])
save_name = 'diabetes PRC curve.png'
plt.savefig(save_name)
plt.show()

for i in range(8):
    print('The best model measured by ',measures[i],'is
',names[np.argmax(scores_table[:7,i])])
csv_name = 'diabetes.csv'
np.savetxt(csv_name, scores_table, delimiter=",")
```

Batch Comparison

The way Sheryl views the dataset consists of the following important parts:

1) See how balanced or unbalanced the data set is.

```
print(" Outcome distribution")
print(diabetes.groupby('Outcome').size())
ratio = diabetes['Outcome'].sum()/(diabetes['Outcome'].sum() + (1-
diabetes['Outcome']).sum())
print(ratio)
1-ratio
```

2) Cleaning the data, replacing the NaN values with the mean or median. Replace 0 values to NaN values. Then sum the null values in each of those features, to know how many null values we have.

```
# We replace the NaN values with the mean or median.
# Glucose and BloodPressure dont have much outliers, and we need little data to
fill. The mean will be enough.
# The others, has a huge disparity between some samples, and we need a lot of
data. So the median is best.
diabetes["Glucose"].fillna(diabetes["Glucose"].mean(),inplace=True)
diabetes["BloodPressure"].fillna(diabetes["BloodPressure"].mean(),inplace=True)
diabetes["SkinThickness"].fillna(diabetes["SkinThickness"].median(),inplace=True
)
diabetes["Insulin"].fillna(diabetes["Insulin"].median(),inplace=True)
diabetes["BMI"].fillna(diabetes["BMI"].median(),inplace=True)

print (diabetes.isnull().sum())
print ('--'*40)
diabetes.info()
print ('--'*40)
diabetes.head()
diabetes.describe()

# Replace 0 values to NaN values. Then sum the null values in each of those
features,
#to know how many null values we have.
diabetes_copy=diabetes.copy(deep=True) ## We will need later the diabetes
dataset with the 0s.

diabetes[["Glucose","BloodPressure","SkinThickness","Insulin","BMI"]]=diabetes[[
"Glucose","BloodPressure","SkinThickness","Insulin","BMI"]].replace(0,np.NaN)
print (diabetes.isnull().sum())
diabetes.describe()
```

3) Pairplot the data.

```
# pair plot for clean data
p=sns.pairplot(diabetes, hue = 'Outcome')
```

4) Plot the heatmap for clean data.

```
# heatmap for clean data
plt.figure(figsize=(12,10))  # on this line I just set the size of figure to 12
by 10.
p=sns.heatmap(diabetes.corr(), annot=True,cmap ='RdYlGn')  # seaborn has very
simple solution for heatmap
```

To compare results, we need to import baseline models and also split the data into training and test sets after we normalize the data.

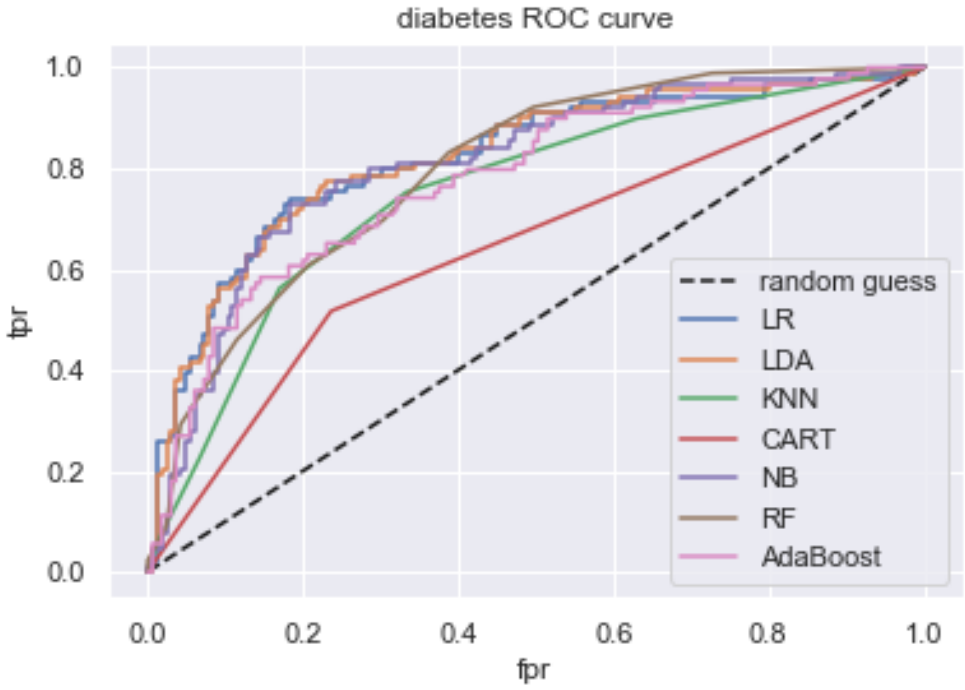
```
# import baseline models
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
```

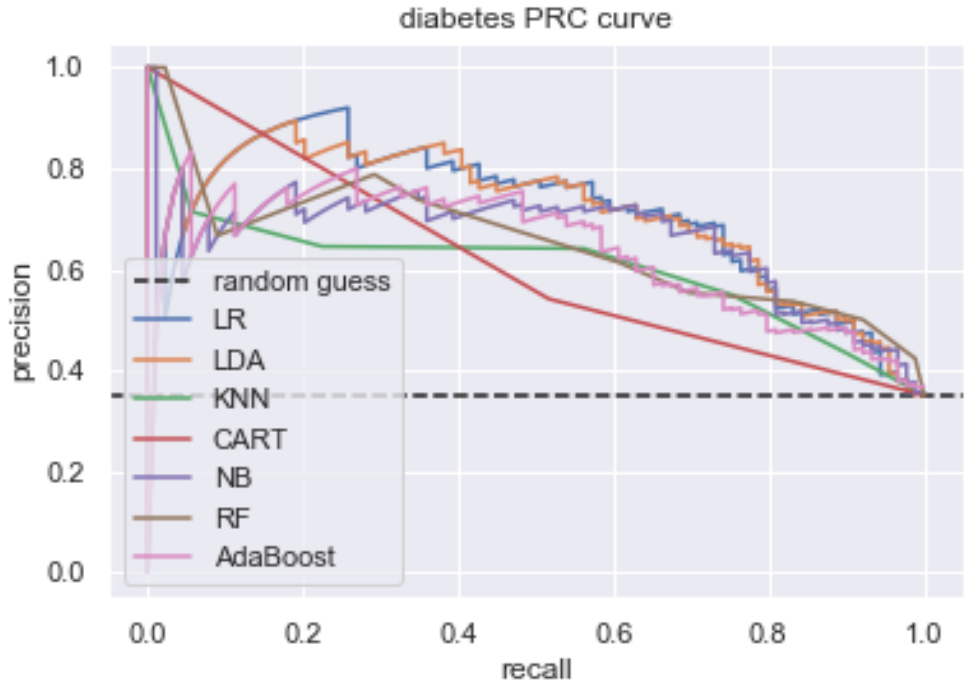
```
from sklearn import model_selection
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import AdaBoostClassifier
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as Pipeline
from sklearn.datasets import make_classification
from sklearn.model_selection import (GridSearchCV,StratifiedKFold)
# scale and split
from sklearn.preprocessing import StandardScaler
X=diabetes_copy.drop(["Outcome"], axis=1)
y=diabetes_copy["Outcome"]
print (X.info())
columnas=
["Pregnancies","Glucose","BloodPressure","SkinThickness","Insulin","BMI","Diabet
esPedigreeFunction","Age"]
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
X=pd.DataFrame(X_scaled, columns=[columnas])
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=444, stratify=y)
```

Batch Comparison Result

The results are given below:

classifier\measure	AUROC	AUPRC	accuracy_	accuracy	average_p	f1	log_loss_s	recall
LogisticRegression	0.82	0.70	0.75	0.78	0.58	0.65	-7.48	0.57
LinearDiscriminantAnalysis	0.82	0.70	0.77	0.78	0.57	0.64	-7.61	0.56
KNeighborsClassifier	0.75	0.60	0.74	0.74	0.51	0.60	-9.11	0.56
DecisionTreeClassifier	0.64	0.61	0.65	0.68	0.45	0.53	-11.15	0.52
GaussianNB	0.81	0.66	0.76	0.78	0.58	0.67	-7.48	0.62
RandomForestClassifier	0.80	0.65	0.73	0.74	0.50	0.55	-9.11	0.46
AdaBoostClassifier	0.78	0.63	0.70	0.76	0.54	0.62	-8.43	0.57
RandomGuess	0.50	0.35	0.65	0.65	0.35			





From the table and comparing graphs, we can see that AUROC for different classifiers may not differ much, but AUPRC tells more difference.

The best model measured by AUROC and accuracy_best_threshold(by ROC) is LDA. The best model measured by AUPRC, accuracy, and log_loss_score is LR. The best model measured by average_precision, f1, and recall is NB.

Therefore, together with the figures and detailed analysis above, AUPRC may be a better measurement than AUROC in this diabetes dataset. Besides, we note that precision, recall and F1 are related to each other, AUROC and accuracy_best_threshold(by ROC) are related to each other, but AUPRC, accuracy and log_loss are not that related. Therefore, LR(Logistic Regression) may be the best model for our dataset.

However, while ROC curves look nice for most classifiers, PRC curves may be quite unstable. As a result, it may be useful to use both for model selection. In other words, when we are choosing classifier types, it may be reasonable to use AUROC, and test the model on baseline algorithms. However, when we are fitting our hyperparameters, or when we are improving our classifier, it may be helpful to turn to AUPRC or the Recall-Precision Curve.