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.

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.

Useful codes

The codes from Sheryl that can generate ROC curve, PRC curve, AUROC, AUPRC, accuracy with best ROC threshold, and save figures as well as give out a numeric report, can be modified from the following:

```
def Find_Optimal_Cutoff(target, predicted):
    """ Find the optimal probability cutoff point for a classification model
related to event rate
    Parameters
    ----------
    target : Matrix with dependent or target data, where rows are observations

    predicted : Matrix with predicted data, where rows are observations

    Returns
    -------
    list type, with optimal cutoff value

    """
    fpr, tpr, threshold = roc_curve(target, predicted)
    i = np.arange(len(tpr))
    roc = pd.DataFrame({'tf' : pd.Series(tpr-(1-fpr), index=i), 'threshold' :
pd.Series(threshold, index=i)})
    roc_t = roc.ix[(roc.tf-0).abs().argsort()[:1]]

    return list(roc_t['threshold'])

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.33, random_state=444, stratify=y)

pipe2 = Pipeline([
```

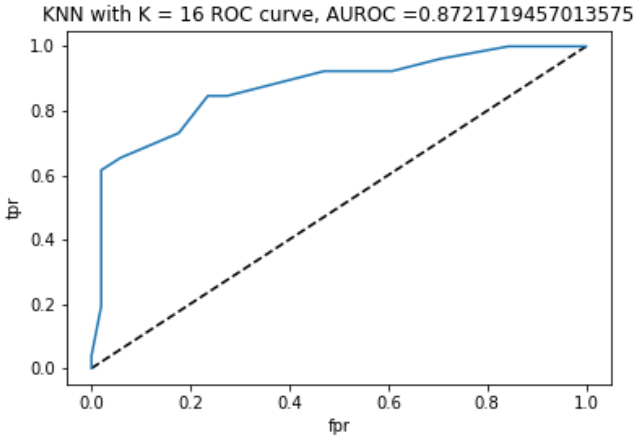
```
    ('oversample', SMOTE(random_state=444)),
    ('clf', LinearDiscriminantAnalysis())])

skf2 = StratifiedKFold(n_splits=10)
param_grid = {'clf__n_components': [1]}
grid = GridSearchCV(pipe2, param_grid, return_train_score=False,
                    n_jobs=-1, scoring="roc_auc", cv=skf2)
LDA=grid.fit(X_train, y_train)
from sklearn.metrics import roc_curve
y_pred_proba =LDA.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
#Area under ROC curve
from sklearn.metrics import roc_auc_score
auroc = roc_auc_score(y_test,y_pred_proba)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='LDA')
plt.xlabel('fpr')
plt.ylabel('tpr')
title_name = 'LDA ROC curve, AUROC ='+str(auroc)
plt.title(title_name)
plt.savefig('LDA ROC curve.png')
plt.show()
print('AUROC = ',auroc)
from sklearn.metrics import precision_recall_curve
precision, recall, thresholds = precision_recall_curve(y_test,y_pred_proba)
from sklearn.metrics import auc
auprc = auc(recall, precision)
plt.plot([1,0],[0,1],'k--')
plt.plot(recall,precision, label='LDA')
plt.xlabel('recall')
plt.ylabel('precision')
title_name = 'LDA PRC curve, AUPRC ='+str(auprc)
plt.title(title_name)
plt.savefig('LDA PRC curve.png')
plt.show()
print('AUPRC = ',auprc)
threshold = Find_Optimal_Cutoff(y_test,y_pred_proba)
from sklearn.metrics import accuracy_score
y_pred = y_pred_proba>threshold
accuracy = accuracy_score(y_test, y_pred)

# report of scores
print('AUROC = ',auroc,', AUPRC = ',auprc,'. Best threshold for ROC =
',threshold[0], ', accuracy is then ',accuracy,'.')
```

KNN with K = 16

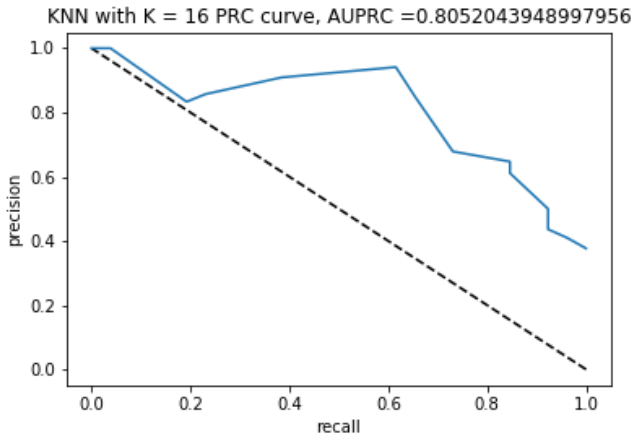
The first classifier is KNN with K = 11.The corresponding jupyter notebook file is pima-diabetes-using-logistic-and-knn-84.ipynb. The file edited by Sheryl is pima-diabetes-using-logistic-and-knn-84-Sheryl1020.ipynb, adding PRC curve and AUPRC, best threshold for ROC, average accuracy, as well as the average precision score.

The scores from the KNN with K = 16 classifier are: AUROC = 0.8721719457013575 , AUPRC = 0.8052043948997956 . Best threshold for ROC = 0.4375 , accuracy is then 0.7922077922077922 .The best threshold is not so close to the proportion of positive data points. Therefore, the result from the KNN classifier with K = 11 is better than the random guess classifier in all the three scores.

A visual result for the ROC curve for the KNN classifier is given in the following figure.



A visual result for the PRC curve for the KNN classifier is given in the following figure.

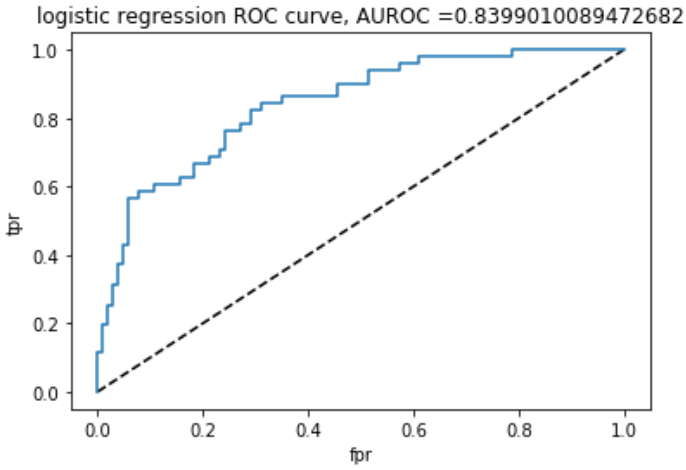


Logistic Regression

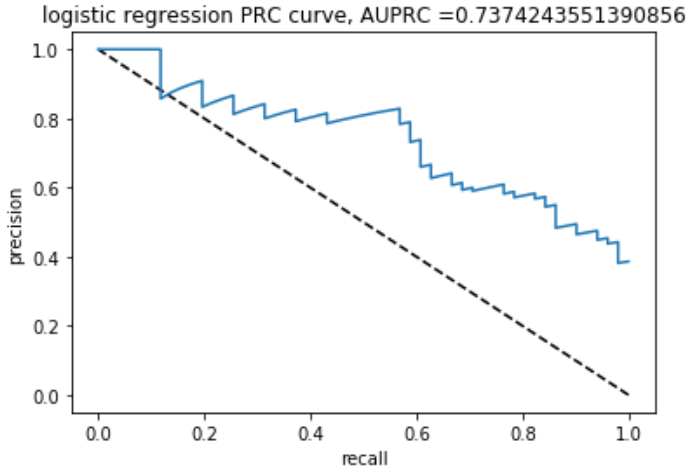
The second classifier is logistic regression. The corresponding jupyter notebook file is pima-diabetes-using-logistic-and-knn-84.ipynb. The file edited by Sheryl is pima-diabetes-using-logistic-and-knn-84-Sheryl1020.ipynb, adding PRC curve and AUPRC, best threshold for ROC, average accuracy, as well as the average precision score.

The scores from this classifier are: AUROC = 0.8399010089472682 , AUPRC = 0.7374243551390856 . Best threshold for ROC = 0.3647386502429486 , accuracy is then 0.7532467532467533 . The best threshold is quite close to the proportion of positive data points.. Therefore, the result from this classifier is better than the random guess classifier in all the three scores.

A visual result for the ROC curve for the logistic classifier is given in the following figure.



A visual result for the PRC curve for the logistic classifier is given in the following figure.

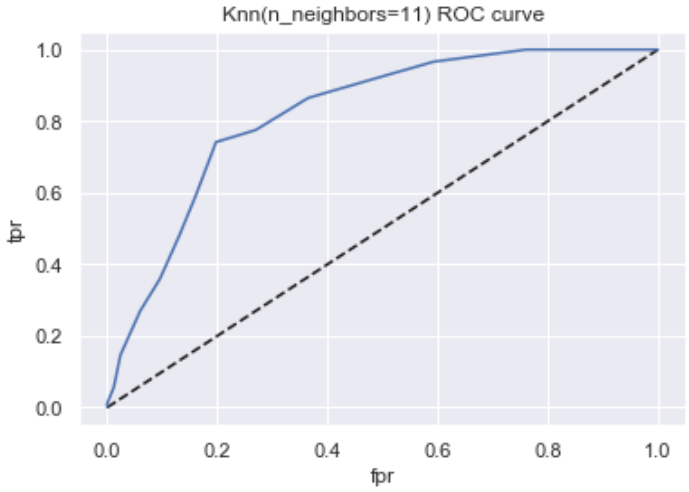


KNN with K = 11

The third classifier is KNN with K = 11. The corresponding jupyter notebook file is step-by-step-diabetes-classification-knn-detailed.ipynb. The file edited by Sheryl for KNN(K =11) is diabetes_Sheryl_1019.ipynb, adding PRC curve and AUPRC, best threshold for ROC, average accuracy, as well as the average precision score.

The KNN model uses risk for validation set to choose K. The scores from the KNN with K = 11 classifier are: AUROC = 0.8215367018771446, AUPRC = 0.6622327480790501. The best threshold is 0.35714285714285715, which is quite close to the proportion of positive data points. With this threshold, the average accuracy is 0.78125. Therefore, the result from the KNN classifier with K = 11 is better than the random guess classifier in all the three scores.

A visual result for the ROC curve for the KNN classifier is given in the following figure.



ROC curve for the KNN classifier, with AUROC = 0.8215367018771446 A visual result for the PRC curve for the KNN classifier is given in the following figure.

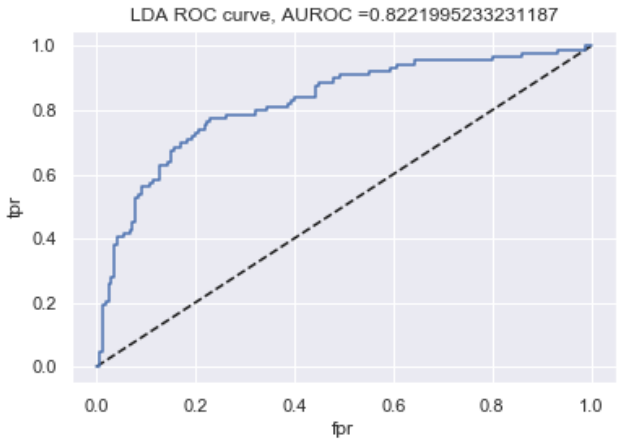
PRC curve for the KNN classifier, with AUPRC = 0.6622327480790501

Linear Discriminant Analysis(LDA)

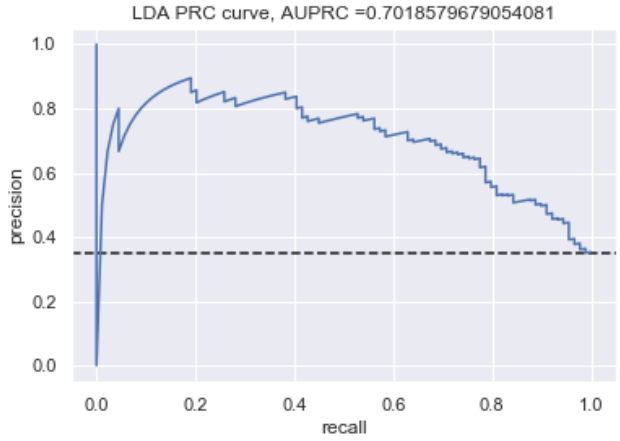
The fourth classifier is Linear Discriminant Analysis(LDA). The corresponding jupyter notebook file is a-complete-ml-work-84-roc-auc.ipynb. The file edited by Sheryl is a-complete-ml-work-84-roc-auc-Sheryl1020.ipynb, adding PRC curve and AUPRC, best threshold for ROC, average accuracy, as well as the average precision score.

The scores from this classifier are: AUROC = 0.8404494382022472 , AUPRC = 0.7040323030097024 . Best threshold for ROC = 0.46555766171966545 , accuracy is then 0.7637795275590551 . The best threshold is not so close to the proportion of positive data points. The result from this classifier is better than the random guess classifier in all the three scores.

A visual result for the ROC curve for the LDA classifier is given in the following figure.



A visual result for the PRC curve for the LDA classifier is given in the following figure.



More Classifiers to Compare

The file edited by Sheryl called a-complete-ml-work-84-roc-auc-Sheryl1020.ipynb contains a comparison for 7 different classifiers and their AUROC, Average Precision(simiar but not the same as AUPRC), Accuracy, with cross validation. The corresponding code is the following:

```
# added by Sheryl, more classifiers
#scoring =  'accuracy'
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(('SVM', SVC()))
models.append(('AdaBoost', AdaBoostClassifier()))

#print("evaluation metric: " + scoring)
results_accuracy=[]
results_auroc=[]
results_average_precision=[]
results_neg_log_loss=[]
names=[]
scores_table = np.zeros([7,4])
i = 0 # looping index
```

```
for name, model in models:
        kfold = model_selection.KFold(n_splits=10, random_state=seed)
        cv_results_accuracy = model_selection.cross_val_score(model,X, y,
cv=kfold, scoring='accuracy')
        results_accuracy.append(cv_results_accuracy)
        cv_results_auroc = model_selection.cross_val_score(model,X, y, cv=kfold,
scoring='roc_auc')
        results_auroc.append(cv_results_auroc)
        cv_results_neg_log_loss = model_selection.cross_val_score(model,X, y,
cv=kfold, scoring='neg_log_loss')
        results_neg_log_loss.append(cv_results_neg_log_loss)
        cv_results_average_precision = model_selection.cross_val_score(model,X,
y, cv=kfold, scoring='average_precision')
        results_average_precision.append(cv_results_average_precision)
        names.append(name)


        # report of scores
        print ("Algorithm :",name)
        print (" Data clean & scaled CV AUROC mean: ", cv_results_auroc.mean())
        print (" Data clean & scaled CV accuracy mean: ",
cv_results_accuracy.mean())
        print (" Data clean & scaled CV average_precision mean: ",
cv_results_average_precision.mean())
        print (" Data clean & scaled CV neg_log_loss mean: ",
cv_results_neg_log_loss.mean())
        scores_table[i, 0] = cv_results_auroc.mean()
        scores_table[i, 1] = cv_results_accuracy.mean()
        scores_table[i, 2] = cv_results_average_precision.mean()
        scores_table[i, 3] = cv_results_neg_log_loss.mean()
        #print('AUROC = ',auroc,', AUPRC = ',auprc,'. Best threshold for ROC =
',threshold[0], ', accuracy is then ',accuracy,'.')
        print ("--"*30)
        i = i + 1

print(scores_table)
np.savetxt("scores_table.csv", scores_table, delimiter=",")
```

The results are given below:

classifier\measure	AUROC	accuracy	AP~AUPRC	neg_log_loss
LogisticRegression	0.8362	0.7747	0.7223	-0.4746
LinearDiscriminantAnalysis	0.8368	0.7696	0.7208	-0.4770
KNeighborsClassifier	0.7940	0.7304	0.6300	-1.6233
DecisionTreeClassifier	0.6757	0.6979	0.4788	-10.4361
GaussianNB	0.8169	0.7487	0.6767	-0.6811
RandomForestClassifier	0.7922	0.7526	0.6424	-0.9474
AdaBoostClassifier	0.8142	0.7461	0.6892	-0.6514
RandomGuess	0.5000	0.6500	0.3500	

From this table, we can see that AUROC for different classifiers may not differ much, but average precision, which is similar to AUPRC, tells more difference. Therefore, together with the figures and detailed analysis above, AUPRC may be a better measurement in this diabetes 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.

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(('SVM', SVC()))
models.append(('AdaBoost', AdaBoostClassifier()))

#print("evaluation metric: " + scoring)
results_accuracy=[]
results_auroc=[]
results_average_precision=[]
results_neg_log_loss=[]
results_f1 = []
results_recall =[]
names=[]
measures =
['AUROC','AUPRC','accuracy_best_threshold','accuracy','average_precision','f1','
log_loss_score','recall']
scores_table = np.zeros([7,8])
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)
        #Area under ROC curve
        auroc = roc_auc_score(y_test,y_pred_proba)
        plt.plot([0,1],[0,1],'k--')
        plt.plot(fpr,tpr, label=name)
        plt.xlabel('fpr')
        plt.ylabel('tpr')
        title_name = name + ' ROC curve, AUROC ='+str(auroc)
        plt.title(title_name)
        save_name = name + ' ROC curve.png'
        plt.savefig(save_name)
        plt.show()
        print('AUROC = ',auroc)
        precision, recall, thresholds =
precision_recall_curve(y_test,y_pred_proba)
        auprc = auc(recall, precision)
        plt.axhline(y=ratio, xmin=0, xmax=1,color='k', linestyle = '--')
        plt.plot(recall,precision, label=name)
```

```
        plt.xlabel('recall')
        plt.ylabel('precision')
        title_name = name + ' PRC curve, AUPRC ='+str(auprc)
        plt.title(title_name)
        save_name = name + ' PRC curve.png'
        plt.savefig(save_name)
        plt.show()
        print('AUPRC = ',auprc)
        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,' for train_test split: AUROC = ',auroc,', AUPRC = ',auprc,',
average precision = ,',average_precision,
              '. \nBest threshold for ROC = ',threshold[0], ', accuracy for the
best ROC threshold is then ',accuracy_best_threshold,', accuracy = ', accuracy,
              '. \nF1 score = ', f1, ', log loss = ',log_loss_score,', recall
=', recall,'.')
        print ("--"*30)
        i = i + 1

print(scores_table)
for i in range(8):
    print('The best model measured by ',measures[i],'is
',names[np.argmax(scores_table[:,i])])

np.savetxt("scores_table1024_new.csv", scores_table, delimiter=",")
```

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
```

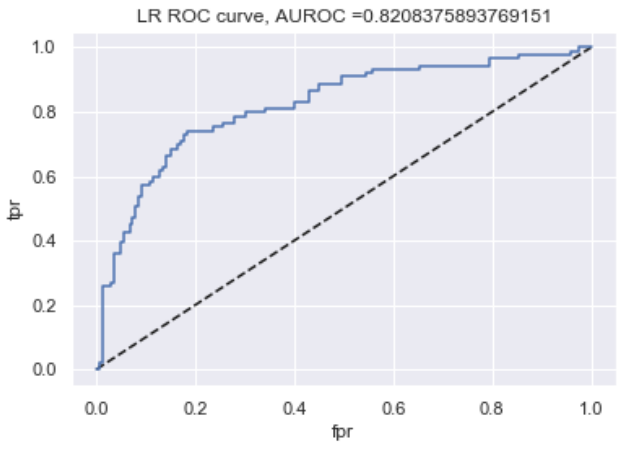
To compare results, we need to import baseline models and also split the data into training and test sets after we normalize the data.

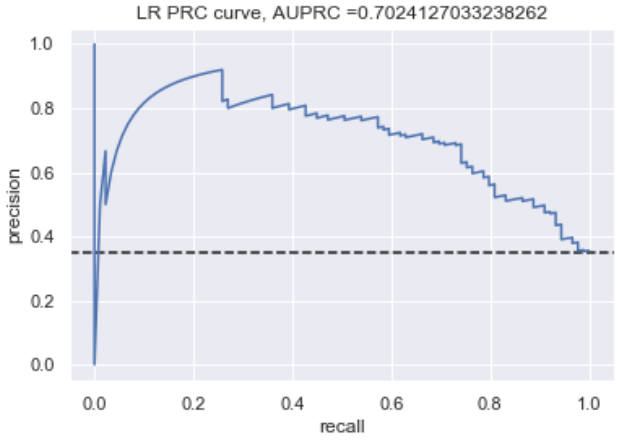
```
# import baseline models
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
```

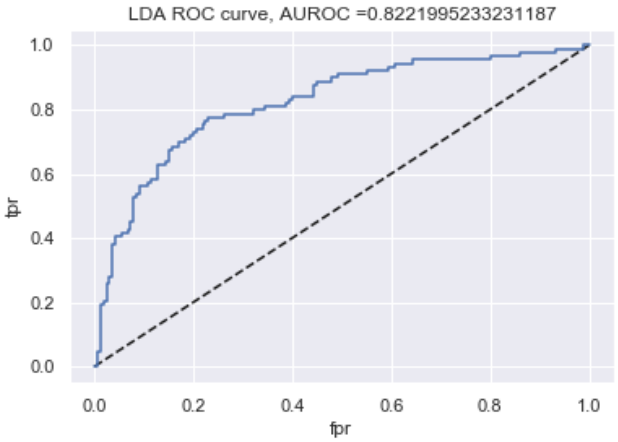
```
from sklearn import model_selection
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
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)
```

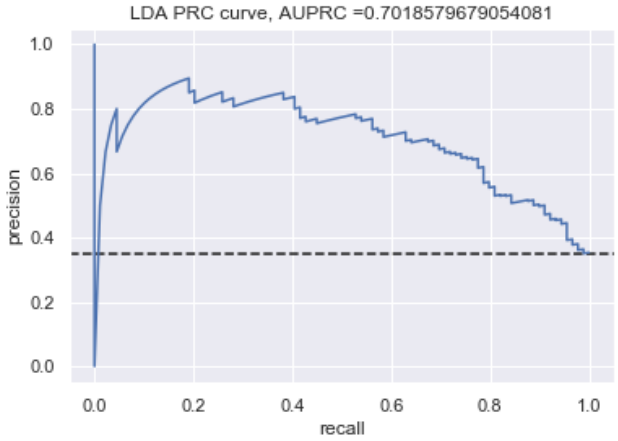
Batch Comparison Result

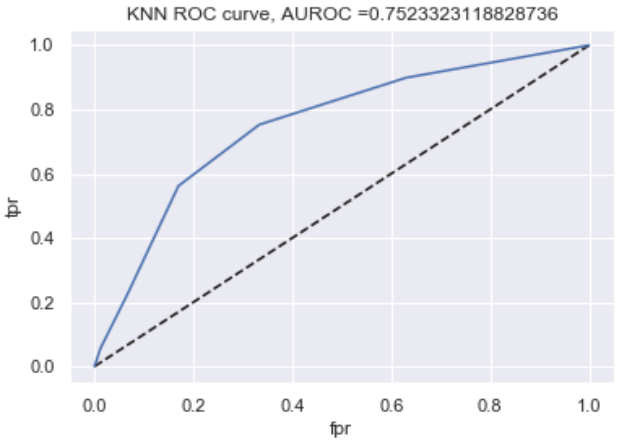
The resulting curves are given below:

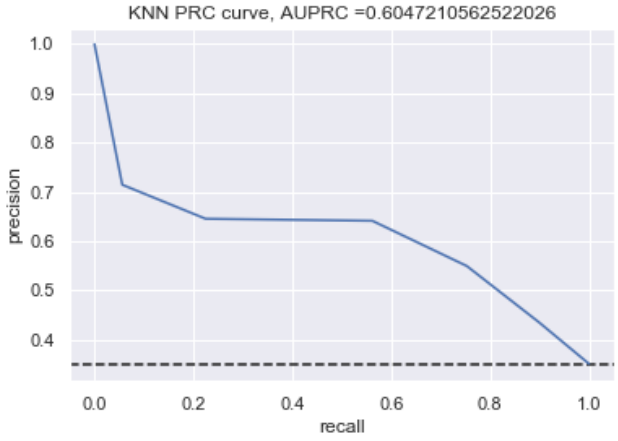


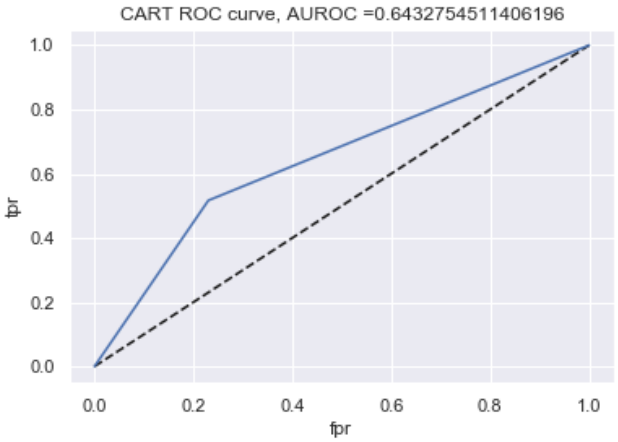

LR PRC curve, AUPRC =0.7024127033238262
precision
recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

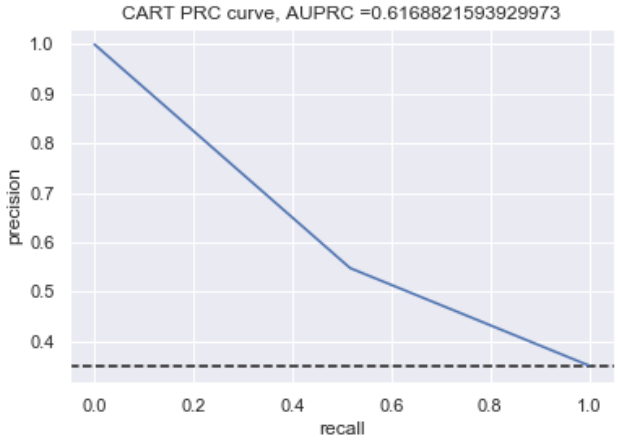

LDA ROC curve, AUROC =0.8221995233231187
tpr
fpr
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

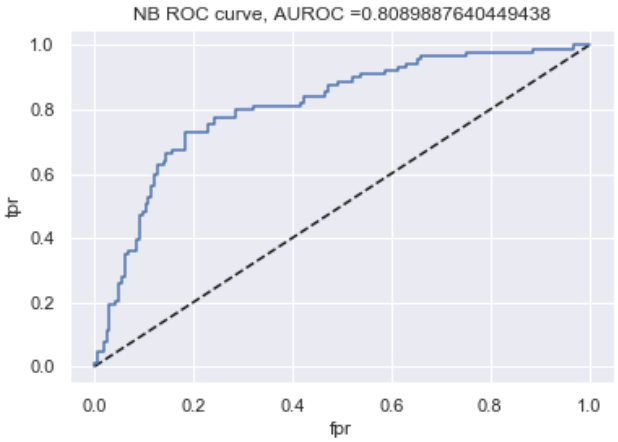

LDA PRC curve, AUPRC =0.7018579679054081
precision
recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

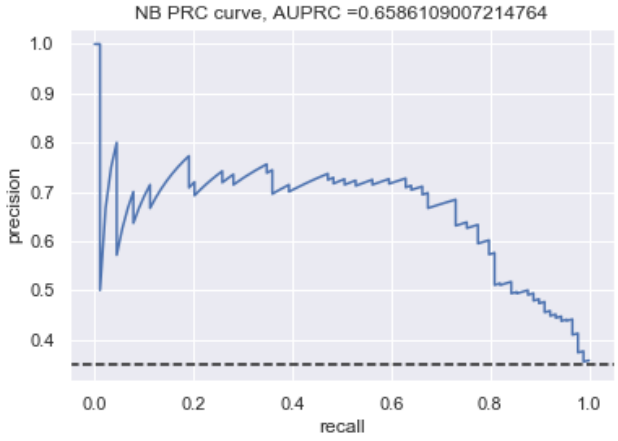

KNN ROC curve, AUROC =0.7523323118828736
tpr
fpr
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

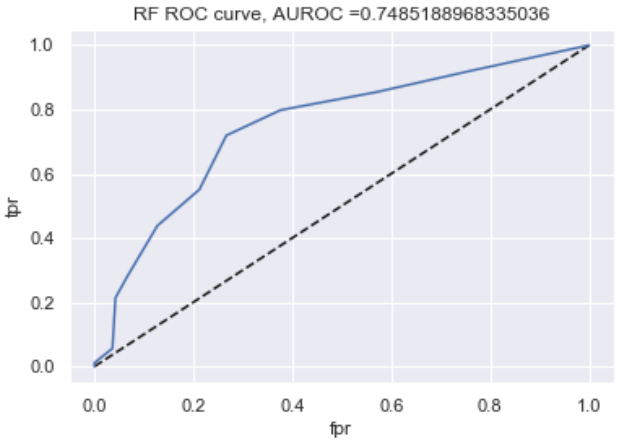

KNN PRC curve, AUPRC =0.6047210562522026
precision
recall
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.0
0.2
0.4
0.6
0.8
1.0

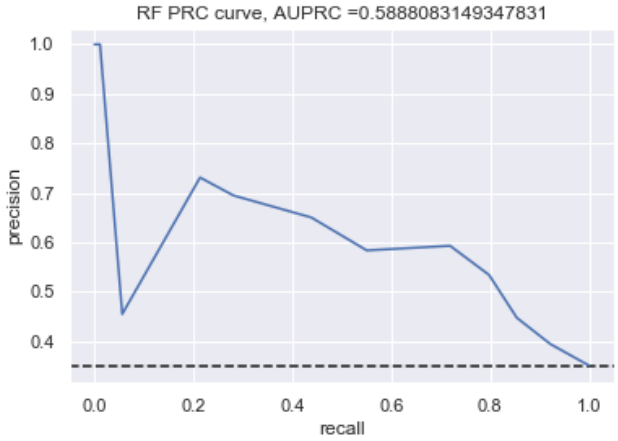

CART ROC curve, AUROC =0.6432754511406196
tpr
fpr
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

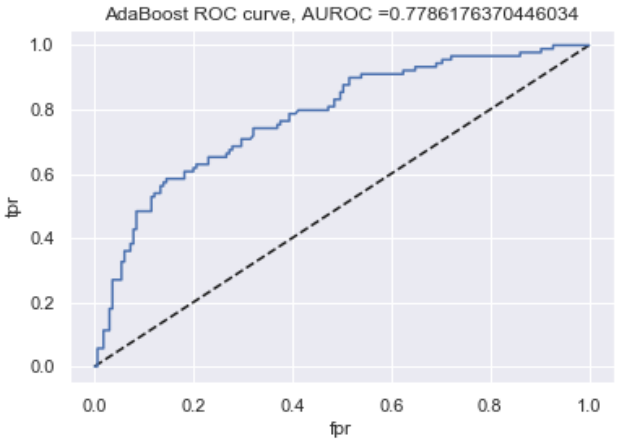

CART PRC curve, AUPRC =0.6168821593929973
precision
recall
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.0
0.2
0.4
0.6
0.8
1.0

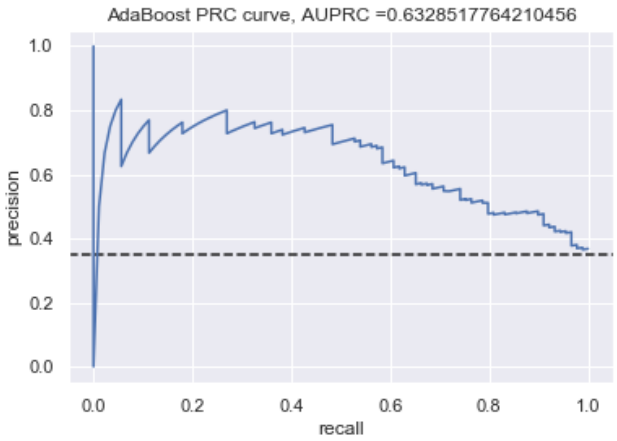

NB ROC curve, AUROC =0.8089887640449438
tpr
fpr
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


NB PRC curve, AUPRC =0.6586109007214764
precision
recall
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.0
0.2
0.4
0.6
0.8
1.0


RF ROC curve, AUROC =0.7485188968335036
tpr
fpr
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


RF PRC curve, AUPRC =0.5888083149347831
precision
recall
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.0
0.2
0.4
0.6
0.8
1.0


AdaBoost ROC curve, AUROC =0.7786176370446034
tpr
fpr
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0



The resulting table is given below:

classifier\measure	AUROC	AUPRC	accuracy_best_threshold	accuracy	average_precision	f1	log_loss_score	recall
LogisticRegression	0.8208	0.7024	0.7520	0.7835	0.5794	0.6497	7.4789	0.5730
LinearDiscriminantAnalysis	0.8222	0.7019	0.7677	0.7795	0.5728	0.6410	7.6149	0.5618
KNeighborsClassifier	0.7523	0.6047	0.7362	0.7362	0.5137	0.5988	9.1107	0.5618
DecisionTreeClassifier	0.6584	0.6342	0.6496	0.7008	0.4702	0.5476	10.3345	0.5169
GaussianNB	0.8090	0.6586	0.7598	0.7835	0.5811	0.6667	7.4789	0.6180
RandomForestClassifier	0.7716	0.6542	0.7283	0.7283	0.4959	0.5548	9.3827	0.4831
AdaBoostClassifier	0.7786	0.6329	0.7008	0.7559	0.5393	0.6220	8.4308	0.5730
RandomGuess	0.5000	0.3490	0.6510	0.6510	0.3490			

From this table, 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.