NAME: FULLNAME
SECTION: NUMBER
CS 5703: Machine Learning Practices

1 Homework 9: Decision Tree Classifiers

1.1 Assignment Overview

Follow the TODOs and read through and understand any provided code. Post any questions you might have to the class Slack. For all plots, make sure all necessary axes and curves are clearly and accurately labeled. Include figure/plot titles appropriately as well.

1.1.1 Task

For this assignment you will be exploring Decision Tree Classifiers to predict potentially fraudulent providers from summary statistics of their filed healthcare claims.

These data were re-configured from a dataset collected for the purpose of detecting Health care Provider Fraud. Total Medicare spending increases exponentially due to fraud in Medicare claims. Healthcare fraud involves health care providers, physicians, patients, and beneficiaries acting in tandum to construct fraudulent claims.

Features
The features are aggregate statistics computed as either the mean or the sum. For the following features, the column represents the average value for the provider's claims:

- InscClaimAmtReimbursed
- DeductibleAmtPaid
- NoOfMonths_PartACov
- NoOfMonths_PartBCov
- IPAnnualReimbursementAmt
- IPAnnualDeductibleAmt
- OPAnnualReimbursementAmt
- OPAnnualDeductibleAmt
- NumPhysiciansSeen
- NumProcedures
- NumDiagnosisClaims

- Age

For the following features, the column represents the total number among the provider's claims:

- ChronicCond_Alzheimer
- ChronicCond_Heartfailure
- ChronicCond_KidneyDisease
- ChronicCond_Cancer
- ChronicCond_ObstrPulmonary
- ChronicCond_Depression
- ChronicCond_Diabetes
- ChronicCond_IschemicHeart
- ChronicCond_Osteoporasis
- ChronicCond_rheumatoidarthritis
- ChronicCond_stroke
- RenalDiseaseIndicator

These data were amalagmated from the HEALTHCARE PROVIDER FRAUD DETECTION ANALYSIS (https://www.kaggle.com/rohitrox/healthcare-provider-fraud-detection-analysis) data set on Kaggle.

1.1.2 Objectives

- Introduction to Decision Trees

1.1.3 General References

- Guide to Jupyter (https://www.datacamp.com/community/tutorials/tutorial-jupyter-notebook)
- Python Built-in Functions (https://docs.python.org/3/library/functions.html)
- Python Data Structures (https://docs.python.org/3/tutorial/datastructures.html)
- Numpy Reference (https://docs.scipy.org/doc/numpy/reference/index.html)
- Numpy Cheat Sheet (https://s3.amazonaws.com/assets.datacamp.com/blog_assets/Numpy_Python_Cheat_Sheet.pdf)
- Summary of matplotlib (https://matplotlib.org/3.1.1/api/pyplot_summary.html)
- DataCamp: Matplotlib (https://www.datacamp.com/community/tutorials/matplotlib-tutorial-python?utm_source=adwords_ppc&utm_campaignid=1565261270&utm_adgroupid=67750485268&utm_device=c&utm_keyword=&utm_match 299261629574:dsa-473406587955&utm_loc_interest_ms=&utm_loc_physical_ms=9026223&gclid=CjwKCAjw_uDsBRAMEiwAaFiHa8xhgCsO9wVcuZPGj fxYtkBLkQ4E_GjSCZFVCqYCGkphoCjucQAvD_BwE)

- [Pandas DataFrames (https://urldefense.proofpoint.com/v2/url?u=https-3A__pandas.pydata.org_pandas-2Ddocs_stable_reference_api_pandas.DataFrame.html&d=DwMD-g&c=qKdtBuuu6dQK9MsRUVJ2DPXW6oayO8fu4TfEHS8sGNk&r=9ngmsG8rSmDSS-O0b_V0gP-nN_33Vr52qbY3KXuDY5k&m=mcOOc8D0knaNNmmnTEo_F_WmT4j6_nUSL_yoPmGlLWQ&s=h7hQjqucR7tZyfZXxnoy3iitlr32YlrqiFy)]
- [Sci-kit Learn Linear Models (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model)]
- [Sci-kit Learn Ensemble Models (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.ensemble)]
- [Sci-kit Learn Metrics (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics)]
- [Sci-kit Learn Model Selection (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection)]
- [Sci-kit Learn Pipelines (https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html)]
- [Sci-kit Learn Preprocessing (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing)]
- [Decision Trees (https://medium.com/machine-learning-101/chapter-3-decision-trees-theory-e7398adac567)]

1.1.4 Hand-In Procedure

- Execute all cells so they are showing correct results
- Notebook (from Jupyter or Colab):
  - Submit this file (.ipynb) to the Gradescope Notebook HW9 dropbox
- Note: there is no need to submit a PDF file or to submit directly to Canvas

In [1]:
```python
%reload_ext autoreload
%autoreload 2
%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re, os, pathlib
import time as timelib

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import confusion_matrix, roc_curve, auc
from sklearn.metrics import log_loss, f1_score, precision_score
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.linear_model import SGDClassifier, LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import DecisionTreeRegressor, export_graphviz
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, OrdinalEncoder

import joblib
import pickle as pkl

# Default figure parameters
plt.rcParams['figure.figsize'] = (6,5)
plt.rcParams['font.size'] = 10
plt.rcParams['legend.fontsize'] = 10
plt.rcParams['xtick.labelsize'] = 10
plt.rcParams['ytick.labelsize'] = 10
plt.rcParams['figure.constrained_layout.use'] = False
plt.rcParams['axes.titlesize'] = 14
plt.rcParams['axes.labelsize'] = 12


plt.style.use('ggplot')
```

```
In [2]: # COLAB ONLY
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
In [3]: # COLAB ONLY
#
# THIS IMPORTS 3 CUSTOM .py FILES
#
# These are the same python files as we used in HW08
#
# If you are running this on a local machine, don't execute this cell
#

# this is a little weird colab doesn't play _super_ nice with local
# python files
# note that this is not programming best practice
exec(open(
    '/content/drive/My Drive/Colab Notebooks/visualize.py', 'r'
).read())
exec(open(
    '/content/drive/My Drive/Colab Notebooks/metrics_plots.py', 'r'
).read())
exec(open(
    '/content/drive/My Drive/Colab Notebooks/pipeline_components.py', 'r'
).read())
```

```
In [4]: # for local runtimes only (e.g., Jupyter)
from visualize import *
from metrics_plots import *
from pipeline_components import *
```

2 LOAD DATA

In [5]:
```
# TODO: set path appropriately.
fname = "/content/drive/My Drive/MLP_2021/datasets/health_provider_fraud.csv"
#fname = "health_provider_fraud.csv"
claims_data = pd.read_csv(fname)
claims_data.shape
```

Out[5]:
```
(5410, 25)
```

In [6]:
```
""" PROVIDED
Display data info
"""
claims_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5410 entries, 0 to 5409
Data columns (total 25 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   Provider                       5410 non-null   object 
 1   PotentialFraud                 5410 non-null   bool   
 2   Age                            5410 non-null   float64
 3   NumPhysiciansSeen              5410 non-null   float64
 4   NumProcedures                  5410 non-null   float64
 5   NumDiagnosisClaims             5410 non-null   float64
 6   InscClaimAmtReimbursed         5410 non-null   float64
 7   DeductibleAmtPaid              5409 non-null   float64
 8   NoOfMonths_PartACov            5410 non-null   float64
 9   NoOfMonths_PartBCov            5410 non-null   float64
 10  IPAnnualReimbursementAmt       5410 non-null   float64
 11  IPAnnualDeductibleAmt          5410 non-null   float64
 12  OPAnnualReimbursementAmt       5410 non-null   float64
 13  OPAnnualDeductibleAmt          5410 non-null   float64
 14  ChronicCond_Alzheimer          5410 non-null   int64  
 15  ChronicCond_Heartfailure       5410 non-null   int64  
 16  ChronicCond_KidneyDisease      5410 non-null   int64  
 17  ChronicCond_Cancer             5410 non-null   int64  
 18  ChronicCond_ObstrPulmonary     5410 non-null   int64  
 19  ChronicCond_Depression         5410 non-null   int64  
 20  ChronicCond_Diabetes           5410 non-null   int64  
 21  ChronicCond_IschemicHeart      5410 non-null   int64  
 22  ChronicCond_Osteoporasis       5410 non-null   int64  
 23  ChronicCond_rheumatoidarthritis  5410 non-null   int64  
 24  ChronicCond_stroke             5410 non-null   int64  
dtypes: bool(1), float64(12), int64(11), object(1)
memory usage: 1019.8+ KB
```

In [7]:

```python
""" PROVIDED
Display the head of the data
"""
claims_data.head()
```

Out[7]:

	Provider	PotentialFraud	Age	NumPhysiciansSeen	NumProcedures	NumDiagnosisClaims	InscClaimAmtReimbursed	Deductible
0	PRV51001	False	78.840000	1.280000	0.120000	3.640000	4185.600000	213
1	PRV51003	True	70.022727	1.181818	0.363636	5.765152	4588.409091	502
2	PRV51004	False	72.161074	1.322148	0.000000	2.751678	350.134228	2
3	PRV51005	True	70.475536	1.209442	0.000000	2.786266	241.124464	3
4	PRV51007	False	69.291667	1.125000	0.013889	3.208333	468.194444	45

5 rows × 25 columns

In [8]:

```python
""" PROVIDED
Display the summary statistics
Make sure you skim this
"""
claims_data.describe()
```

Out[8]:

	Age	NumPhysiciansSeen	NumProcedures	NumDiagnosisClaims	InscClaimAmtReimbursed	DeductibleAmtPaid	NoOfMonths
count	5410.000000	5410.000000	5410.000000	5410.000000	5410.000000	5409.000000	5
mean	73.731027	1.227410	0.108011	3.676631	1740.679369	155.643175	
std	4.712307	0.220822	0.246305	1.882603	3484.473124	306.489453	
min	34.000000	0.500000	0.000000	0.000000	0.000000	0.000000	
25%	71.768368	1.000000	0.000000	2.696134	232.394593	0.312500	
50%	73.863636	1.200000	0.000000	3.000000	356.085106	4.285714	
75%	75.760000	1.375000	0.083333	3.847902	1490.154301	137.418605	
max	101.000000	3.000000	3.000000	11.000000	57000.000000	1068.000000	

8 rows × 23 columns

3 PRE-PROCESS DATA

In [9]:
```
""" PROVIDED
Construct preprocessing pipeline
"""
selected_features = claims_data.columns.drop(['Provider'])
scaled_features = ['InscClaimAmtReimbursed', 'DeductibleAmtPaid',
                   'IPAnnualReimbursementAmt', 'IPAnnualDeductibleAmt',
                   'OPAnnualReimbursementAmt', 'OPAnnualDeductibleAmt']

pipe = Pipeline([
    ('RowDropper', DataSampleDropper()),
    ('FeatureSelector', DataFrameSelector(selected_features)),
    ('Scale', DataScaler(scaled_features))
])
```

In [10]:
```
""" Provided: execute cell
Pre-process the data using the defined pipeline
"""
processed_data = pipe.fit_transform(claims_data)
processed_data.shape
```

Out[10]:
```
(5409, 24)
```

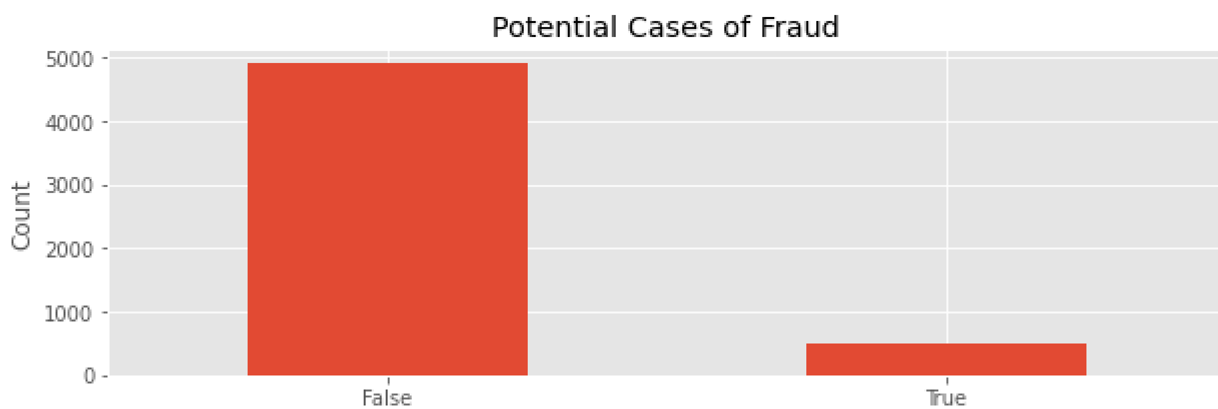
In [11]:
```python
""" PROVIDED: execute cell
Verify all NaNs removed
"""
processed_data.isna().sum()
```

Out[11]:
```
PotentialFraud                    0
Age                               0
NumPhysiciansSeen                 0
NumProcedures                     0
NumDiagnosisClaims                0
InscClaimAmtReimbursed            0
DeductibleAmtPaid                 0
NoOfMonths_PartACov               0
NoOfMonths_PartBCov               0
IPAnnualReimbursementAmt          0
IPAnnualDeductibleAmt             0
OPAnnualReimbursementAmt          0
OPAnnualDeductibleAmt             0
ChronicCond_Alzheimer             0
ChronicCond_Heartfailure          0
ChronicCond_KidneyDisease         0
ChronicCond_Cancer                0
ChronicCond_ObstrPulmonary        0
ChronicCond_Depression            0
ChronicCond_Diabetes              0
ChronicCond_IschemicHeart         0
ChronicCond_Osteoporasis          0
ChronicCond_rheumatoidarthritis   0
ChronicCond_stroke                0
dtype: int64
```

4 VISUALIZE DATA

In [12]:
```python
""" PROVIDED
Plot the class distributions for no potential fraud and potential fraud
"""
class_counts = pd.value_counts(processed_data['PotentialFraud'])
class_counts.plot(kind='bar', rot=0, figsize=(10,3))
plt.title("Potential Cases of Fraud")
plt.ylabel("Count")

# Display the class fractions
nsamples, nfeatures = processed_data.shape
class_counts / nsamples
```

Out[12]:
```
False    0.906452
True     0.093548
Name: PotentialFraud, dtype: float64
```



In [13]:
```python
""" PROVIDED
Extract indices of the postive and negative cases
"""
pos = processed_data['PotentialFraud'] == 1
neg = processed_data['PotentialFraud'] == 0
```

5 Decision Tree Classifiers

5.0.1 Model Exploration

In [14]:
```
""" PROVIDED
Split data into X (the inputs) and y (the outputs)

Hold out a subset of the data, before training and cross validation
using train_test_split, with stratify equal to something other than NONE,
and a test_size fraction of .2.

For this exploratory section, the held out set of data is a validation set.
For the GridSearch section, the held out set of data is a test set.
"""
targetnames = ['NonFraud', 'Fraud']

# Create the inputs and outputs
X = processed_data.drop(['PotentialFraud'], axis=1).copy()
y = processed_data['PotentialFraud'].values.ravel()

# Split data into train and test sets
Xtrain, Xval, ytrain, yval = train_test_split(X, y, stratify=y, random_state=1138, test_size=0.5)
Xtrain.shape, Xval.shape, ytrain.shape, yval.shape
```

Out[14]:
```
((2704, 23), (2705, 23), (2704,), (2705,))
```

In [15]:
```
""" TODO
Explore interesting hyper-parameters. Train multiple decision trees using the training set only.
Pick your favorite model to leave within your submitted report.
"""
# TODO: Create and fit the model
tree_model = DecisionTreeClassifier(criterion='gini', max_depth = 4)
tree_model.fit(Xtrain, ytrain)
```

Out[15]:
```
DecisionTreeClassifier(max_depth=4)
```

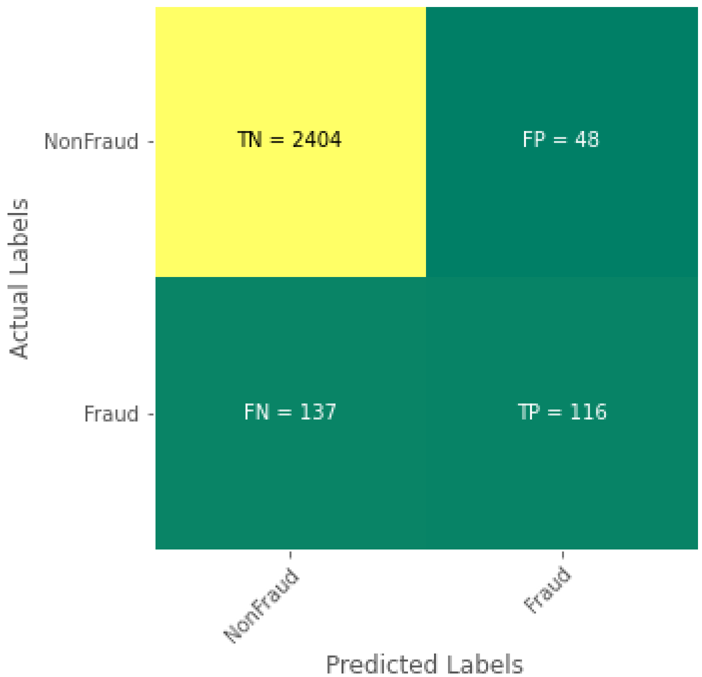
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [16]:
```python
# PROVIDED: Predict with the model on the validation set
preds_val = tree_model.predict(Xval)

# Obtain prediction probabilities for the test set, using
proba_val = tree_model.predict_proba(Xval)

# Obtain the classifier accuracy score for the test set using the
scores = tree_model.score(Xval, yval)

scores
```

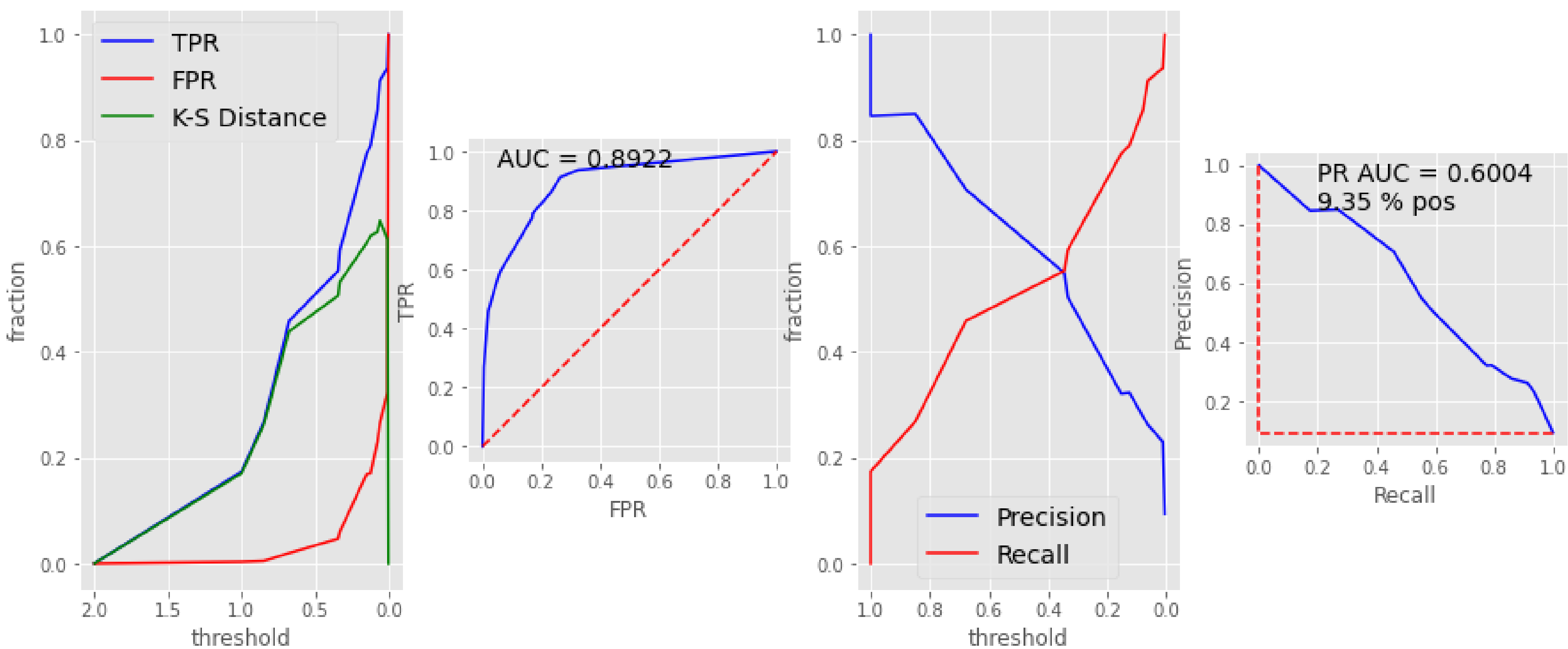
Out[16]:
```
0.9316081330868762
```

In [17]:
```python
""" PROVIDED
Display the confusion matrix, KS plot, ROC curve, and PR curve for the validation set
using metrics_plots.ks_roc_prc_plot

The red dashed line in the PRC is indicative of the expected performance for a random
classifier, which would predict postives at the rate of occurance within the data set
"""
# Confusion Matrix
cmtx_val = confusion_matrix(yval, preds_val)
confusion_mtx_colormap(cmtx_val, targetnames, targetnames)

# Curves
# Note, you'll want the probability class predictions for the class label 1
# See the API page for the DecisionTreeClassifier predict_proba; proba_val[:,1]
roc_prc_results_val  = ks_roc_prc_plot(yval, proba_val[:,1])

# Obtain the PSS and F1 Score
pss_val = skillScore(yval, preds_val)

# pss_val = metrics_plots.skillScore(ytest, preds_val)
f1_val = f1_score(yval, preds_val)
print("PSS: %.4f" % pss_val[0])
print("F1 Score %.4f" % f1_val)
```

```
PSS: 0.4389
F1 Score 0.5564
```



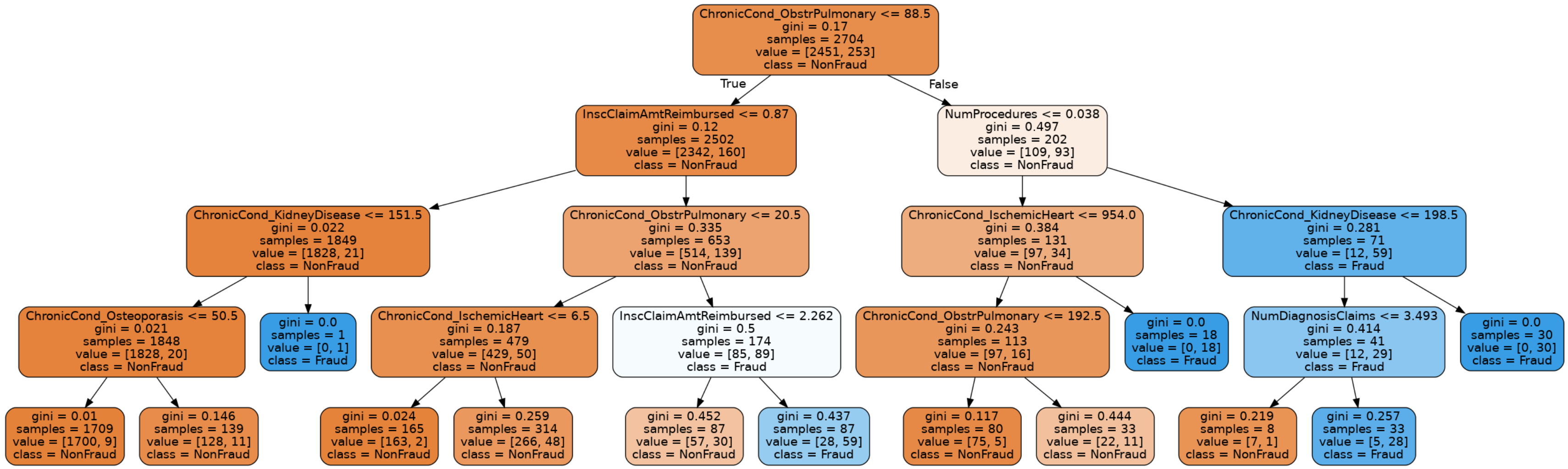
TN = 2404
FP = 48
FN = 137
TP = 116
NonFraud
Fraud
Actual Labels
NonFraud
Fraud
Predicted Labels



TPR
FPR
K-S Distance
fraction
threshold
AUC = 0.8922
TPR
FPR
Precision
Recall
fraction
threshold
PR AUC = 0.6004
9.35 % pos
Precision
Recall

In [18]:
```python
""" PROVIDED
Export the image of the tree model
"""
from IPython.display import Image
export_graphviz(tree_model, out_file='exploratory_model.dot',
                feature_names=X.columns, class_names=targetnames,
                rounded=True, filled=True)
!dot -Tpng exploratory_model.dot > e_model.png
Image(filename='e_model.png')
```

Out[18]:



6 GRID SEARCH CV

In [19]:
```python
def remove_duplicates(arr):
    '''
    Remove duplicates from an array
    '''
    out = []
    for i in arr:
        if not i in out:
            out.append(i)
    return out
```

In [20]:
```python
""" TODO
Set up and run the grid search using GridSearchCV and the following
settings:
* The below scoring dictionary for scoring
* refit set to 'f1' as the optimized metric
* Choose a range of regularization types and parameters
"""
# Optimized metric
opt_metric = 'f1'
scoring = {opt_metric:opt_metric}

# Flag to re-load previous run regardless of whether the file exists
#force = False
force = True

# File previous run is saved to
srchfname = "/content/drive/My Drive/Colab Notebooks/hw9_search_sol_" + opt_metric + ".pkl"
#srchfname = "hw9_search_sol_" + opt_metric + ".pkl"


# SETUP EXPERIMENT HYPERPARAMETERS
# TODO
criterion = ["entropy", "gini"]
max_depth = [2,3,4,5,6,7,8,9,10,12]

# TODO: Create the dictionary of hyper-parameters to try
hyperparams = {"criterion": criterion,
              "max_depth": max_depth}


# RUN EXPERIMENT
time0 = timelib.time()
search = None
if force or (not os.path.exists(srchfname)):
    # Create the GridSearchCV object
    base_model = DecisionTreeClassifier()
    search = GridSearchCV(base_model, hyperparams, scoring=scoring, refit=opt_metric,
                          cv=40, n_jobs=-1, verbose=2, return_train_score=True)

    # TODO: Execute the grid search by calling fit using the training data
    search.fit(Xtrain, ytrain)
```

```python
    # Save the grid search object
    joblib.dump(search, srchfname)
    print("Saved %s" % srchfname)
else:
    # TODO: Re-load the grid search object
    search = joblib.load(srchfname)
    print("Loaded %s" % srchfname)

time1 = timelib.time()
duration = time1 - time0
print("Elapsed Time: %.2f min" % (duration / 60))

search
```

```
Fitting 40 folds for each of 20 candidates, totalling 800 fits
[CV] END .....................criterion=entropy, max_depth=2; total time=   0.0s
[CV] END .....................criterion=entropy, max_depth=2; total time=   0.0s
[CV] END .....................criterion=entropy, max_depth=2; total time=   0.0s
[CV] END .....................criterion=entropy, max_depth=2; total time=   0.0s
[CV] END .....................criterion=entropy, max_depth=2; total time=   0.0s
[CV] END .....................criterion=entropy, max_depth=2; total time=   0.0s
[CV] END .....................criterion=entropy, max_depth=2; total time=   0.0s
[CV] END .....................criterion=entropy, max_depth=2; total time=   0.0s
[CV] END .....................criterion=entropy, max_depth=2; total time=   0.0s
[CV] END .....................criterion=entropy, max_depth=2; total time=   0.0s
[CV] END .....................criterion=entropy, max_depth=2; total time=   0.0s
[CV] END .....................criterion=entropy, max_depth=3; total time=   0.0s
[CV] END .....................criterion=entropy, max_depth=3; total time=   0.0s
[CV] END .....................criterion=entropy, max_depth=3; total time=   0.0s
[CV] END .....................criterion=entropy, max_depth=3; total time=   0.0s
[CV] END .....................criterion=entropy, max_depth=3; total time=   0.0s
[CV] END .....................criterion=entropy, max_depth=3; total time=   0.0s
[CV] END .....................criterion=entropy, max_depth=4; total time=   0.0s
```

7 RESULTS

In [21]:
```python
""" PROVIDED
Display the head of the results for the grid search
See the cv_results_ attribute
"""
all_results = search.cv_results_
df_res = pd.DataFrame(all_results)
df_res.head(3)
```

Out[21]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_criterion	param_max_depth	params	split0_test_f1	split1_tes
0	0.014295	0.002179	0.002964	0.000400	entropy	2	{'criterion': 'entropy', 'max_depth': 2}	0.285714	0.000
1	0.019337	0.003623	0.002882	0.000425	entropy	3	{'criterion': 'entropy', 'max_depth': 3}	0.769231	0.461
2	0.022259	0.002629	0.002835	0.000718	entropy	4	{'criterion': 'entropy', 'max_depth': 4}	0.769231	0.500

3 rows × 92 columns

In [22]:
```python
""" PROVIDE
Obtain the best model from the grid search and
fit it to the full training data
"""
best_model = search.best_estimator_
best_model.fit(Xtrain, ytrain)
```

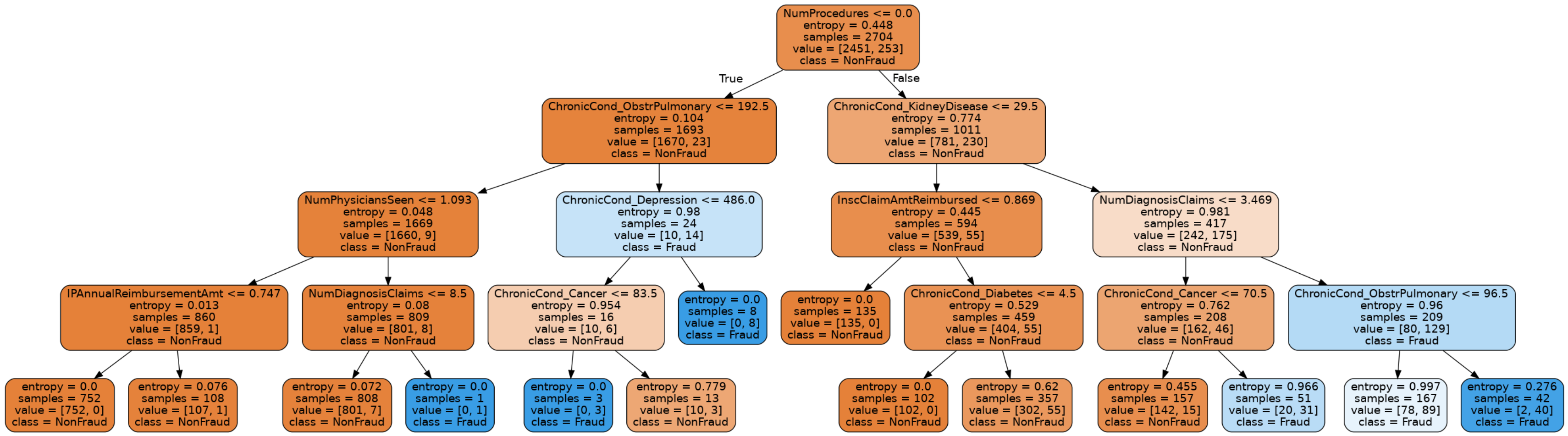
Out[22]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [23]:

```
""" PROVIDED
Export the image of the best model
use export_graphviz
"""
export_graphviz(best_model, out_file='best_model.dot',
                feature_names=X.columns, class_names=targetnames,
                rounded=True, filled=True)
!dot -Tpng best_model.dot > b_model.png
Image(filename='b_model.png')
```

Out[23]:

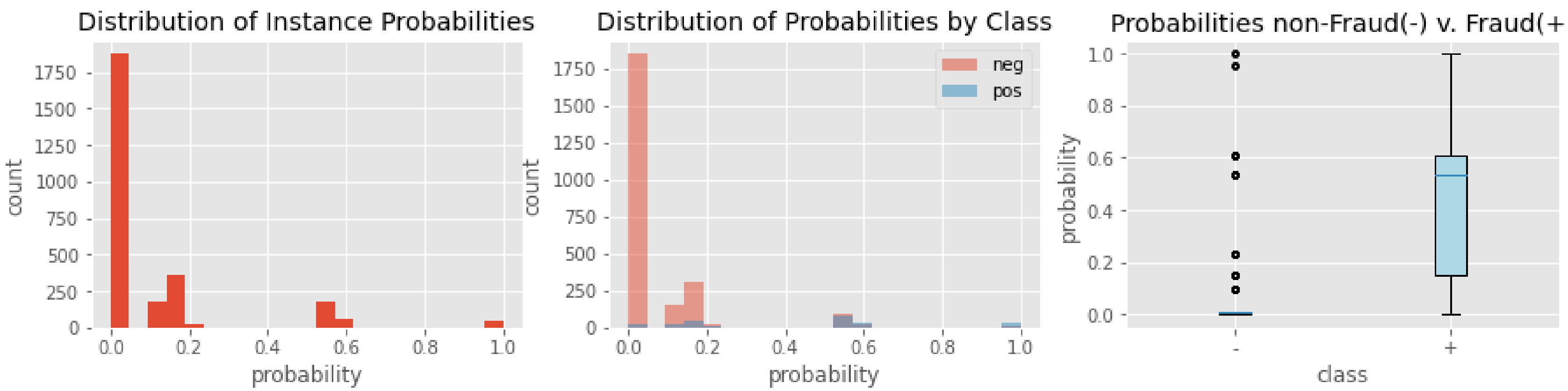


In [24]:

```python
""" PROVIDED
Plot a histogram of the val scores from the best model.
Compare the distribution of probabilities for positive and negative examples
using boxplots.

Create one subplot of the distribution of all the probabilities, with a histogram.
Create a second subplot comparing the distribution of the scores of the
positive examples with the distribution of the negative examples, with boxplots.
"""
# Obtain the pos and neg indices
pos_inds = yval == 1
neg_inds = yval == 0

# Obtain prediction probabilities for the test set (use model.predict_proba)
proba_val = best_model.predict_proba(Xval)

# Separate the probabilities for the pos and neg examples
proba_pos = proba_val[pos_inds, 1]
proba_neg = proba_val[neg_inds, 1]

# Plot the distribution of all probabilities
nbins = 21
plt.figure(figsize=(15,3))
plt.subplot(1,3,1)
plt.hist(proba_val[:,1], bins=nbins)
plt.xlabel('probability')
plt.ylabel('count')
plt.title("Distribution of Instance Probabilities")

plt.subplot(1,3,2)
plt.hist(proba_neg, bins=nbins, alpha=.5)
plt.hist(proba_pos, bins=nbins, alpha=.5)
plt.xlabel('probability')
plt.ylabel('count')
plt.title("Distribution of Probabilities by Class")
plt.legend(['neg', 'pos'])

# Plot the boxplots of the pos and neg examples
plt.subplot(1,3,3)
boxplot = plt.boxplot([proba_neg, proba_pos], patch_artist=True, sym='.')
boxplot['boxes'][0].set_facecolor('pink')
boxplot['boxes'][1].set_facecolor('lightblue')
```

```python
plt.xticks(ticks=[1, 2], labels=['-', '+'])
plt.xlabel("class")
plt.ylabel("probability")
plt.title("Probabilities non-Fraud(-) v. Fraud(+)")
```

Out[24]: Text(0.5, 1.0, 'Probabilities non-Fraud(-) v. Fraud(+)')



7.1 Compare Benchmark to GridSearchCV Best Model

In [25]:
```python
tree_model
```

Out[25]: DecisionTreeClassifier(max_depth=4)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [26]:
```
best_model
```

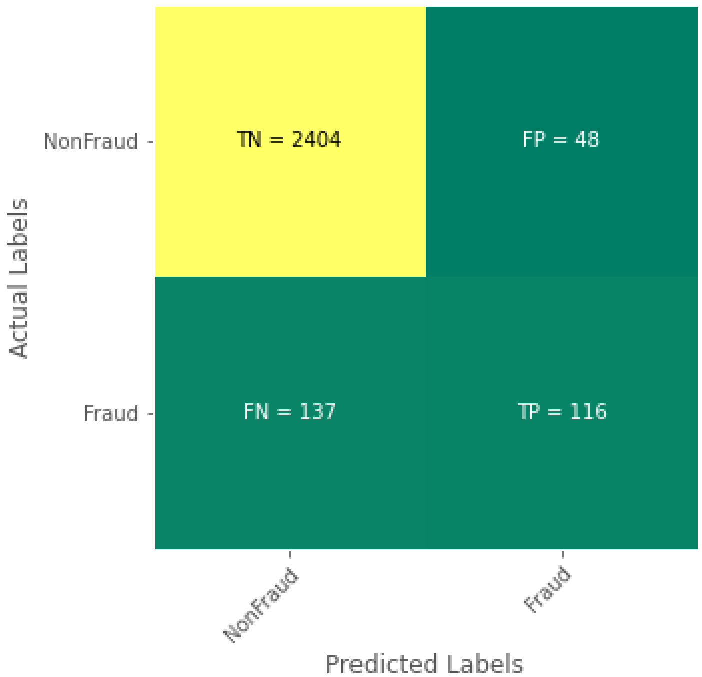
Out[26]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4)
```

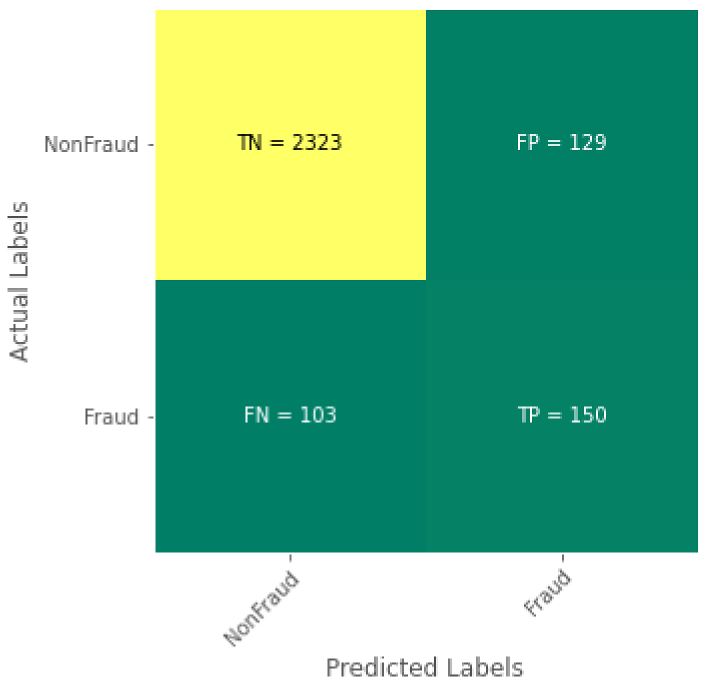
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [27]:
```
# PROVIDED

# Predict with the benchmark model on the validation set
preds_val_bench = tree_model.predict(Xval)

# Predict with the best model on the test set
preds_val_best = best_model.predict(Xval)

# Obtain prediction probabilities for the benchmark model on val set
proba_val_bench = tree_model.predict_proba(Xval)

# Obtain prediction probabilities for the best model on test set
proba_val_best = best_model.predict_proba(Xval)
```

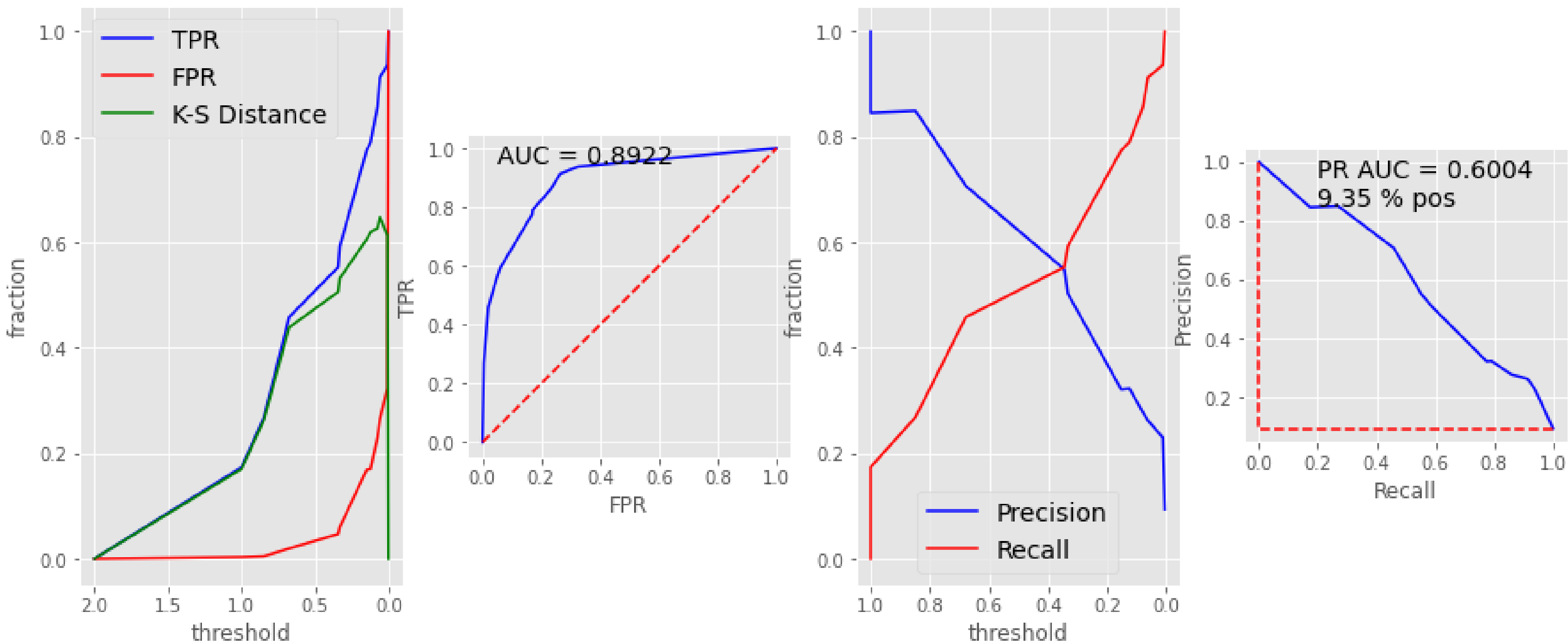
In [28]:
```python
# PROVIDED

# Benchmark tree model validation set confusion matrix
cmtx_val_bench = confusion_matrix(yval, preds_val_bench)
confusion_mtx_colormap(cmtx_val_bench, targetnames, targetnames)

# Best tree model test set confusion matrix
cmtx_val_best = confusion_matrix(yval, preds_val_best)
confusion_mtx_colormap(cmtx_val_best, targetnames, targetnames)
```

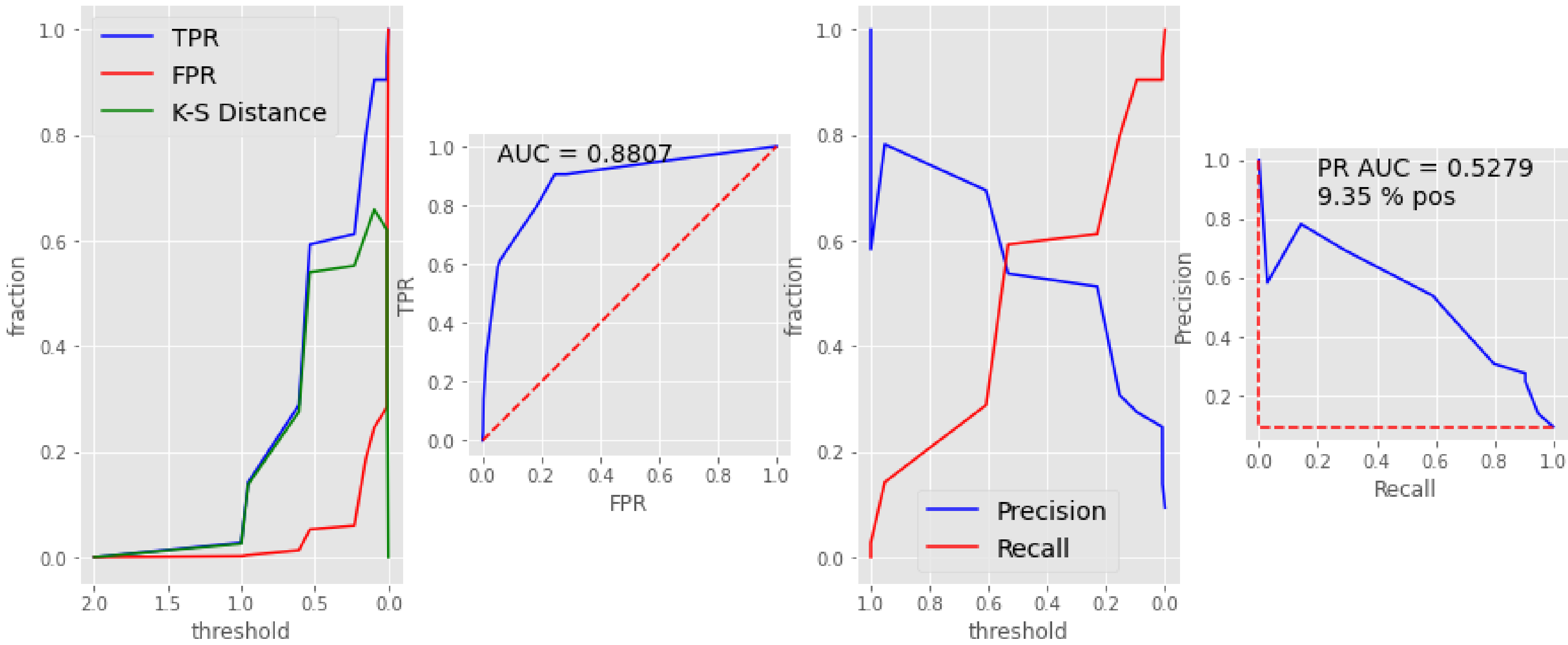
Out[28]:
```
(<Figure size 432x360 with 1 Axes>,
 <AxesSubplot:xlabel='Predicted Labels', ylabel='Actual Labels'>)
```




TN = 2323
FP = 129
FN = 103
TP = 150
NonFraud
Fraud
Actual Labels
NonFraud
Fraud
Predicted Labels

In [29]:
```
# PROVIDED
# Curves (i.e. ROC, PRC, etc) use metrics_plots.ks_roc_prc_plot and the
# the probabilities for the class label of 1

# Benchmark tree validation set performance
roc_prc_results_val_bench  = ks_roc_prc_plot(yval, proba_val_bench[:,1])

# Best tree model validation set performance
roc_prc_results_val_best  = ks_roc_prc_plot(yval, proba_val_best[:,1])
```





8 Discussion

1. Discuss the difference in AUC between your hand-selected model and the best model found by GridSearch

   Answer: The hand-selected model AUC is more better than best model found by GridSearch
2. How many different hyper-parameter sets did GridSearch consider?

   Answer: there is **20** different hyper-parameter sets
3. What was the best set of hyper-parameters according to the GridSearch?

   Answer: the best set of hyper-parameters according to the GridSearch was **{'criterion': 'entropy', 'max_depth': 4}**
4. Discuss the difference in PR-AUC between your hand-selected model and the best model found by GridSearch

   Answer: PR AUC hand-selected model better than model found by GridSearch
5. Examining the learned trees for both models, which features appear to be most important in performing this classification task?

   Answer: the most important in performing this classification task are **ChronicCond_ObstrPulmonary**
6. Relative to the validation data set, what is the best probability threshold to use to distinguish positive from negative classes? Why?

Answer:the best probability threshold to use to distinguish positive from negative classes is **0.1** because even though the accuracy is worse but it can detect about 90% of fraud besides that the negative class with a probability of more than 0.1 can be considered as an outlier as seen in the post and neg example boxplots