homework10-skel

December 3, 2020

NAME: Nigel Mansell
SECTION: 995
CS 5970: Machine Learning Practices

1 Homework 10: FORESTS AND BOOSTING

1.1 Assignment Overview

Follow the TODOs and read through and understand any provided code.
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 Random Forests and Boosting.

1.1.2 Data set

The dataset is based on cyclone weather data from NOAA.
You can obtain the data from the server and git under datasets/cyclones.

We will be predicting whether a cyclone status is a tropical depression (TD) or not.
Status can be the following types:
* TD – tropical depression
* TS – tropical storm
* HU – hurricane intensity
* EX – Extratropical cyclone
* SD – subtropical depression intensity
* SS – subtropical storm intensity
* LO – low, neither a tropical, subtropical, nor extratropical cyclone
* WV – Tropical Wave
* DB – Disturbance

1.1.3 Objectives

- DecisionTreeClassifiers
- RandomForests
- Boosting

1.1.4 Notes

- Do not save work within the mlp_2020 folder

1.1.5 General References

- Guide to Jupyter
- Python Built-in Functions
- Python Data Structures
- Numpy Reference
- Numpy Cheat Sheet
- Summary of matplotlib
- DataCamp: Matplotlib
- Pandas DataFrames
- Sci-kit Learn Trees
- Sci-kit Learn Ensemble Models
- Sci-kit Learn Metrics
- Sci-kit Learn Model Selection
- Sci-kit Learn Pipelines
- Sci-kit Learn Preprocessing

1.1.6 Hand-In Procedure

- Execute all cells so they are showing correct results
- Notebook:
  - Submit this file (.ipynb) to the Canvas HW10 dropbox
- PDF:
  - File/Print/Print to file -> Produces a copy of the notebook in PDF format
  - Submit the PDF file to the Gradescope HW10 dropbox

[1]:
```
# Make sure to have these three custom python files in your
# hw10 working directory
import visualize
import metrics_plots
from pipeline_components import DataSampleDropper, DataFrameSelector
from pipeline_components import DataScaler, DataLabelEncoder

import pandas as pd
import numpy as np
import seaborn as sns
import scipy.stats as stats
import os, re, fnmatch
import pathlib, itertools, time
import matplotlib.pyplot as plt
import matplotlib.patheffects as peffects
import matplotlib.image as mpimg
import time as timelib

from math import ceil, floor
```

```
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.preprocessing import StandardScaler, PolynomialFeatures,␣
↪LabelEncoder
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.model_selection import train_test_split
from sklearn.metrics import explained_variance_score, confusion_matrix
from sklearn.metrics import f1_score, mean_squared_error, roc_curve, auc
import joblib

from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,␣
↪GradientBoostingClassifier

FIGW = 5
FIGH = 5
FONTSIZE = 12

plt.rcParams['figure.figsize'] = (FIGW, FIGH)
plt.rcParams['font.size'] = FONTSIZE

plt.rcParams['xtick.labelsize'] = FONTSIZE
plt.rcParams['ytick.labelsize'] = FONTSIZE

%matplotlib inline
plt.style.use('ggplot')
```

[2]:
```
"""
Display current working directory of this notebook. If you are using
relative paths for your data, then it needs to be relative to the CWD.
"""
HOME_DIR = pathlib.Path.home()
pathlib.Path.cwd()
```

[2]: PosixPath('/home/nigel/Desktop/mlp/homework10')

2 LOAD DATA

[3]:
```
# TODO: set appropriately
filename = '/home/nigel/Desktop/mlp/homework10/cyclones/atlantic.csv'

cyclones_full = pd.read_csv(filename)
nRows, nCols = cyclones_full.shape
print(f'{nRows} rows and {nCols} columns')
```

49105 rows and 22 columns

[4]:
```
""" PROVIDED
not tropical depression (negative case = 0)
is tropical depression (positive case = 1)
"""
targetnames = ['notTropDepress', 'isTropDrepress']

# Determine the positve count
isTD = cyclones_full['Status'].str.contains('TD')
cyclones_full['isTD'] = isTD
npos = isTD.sum()
nneg = nRows - npos

# Compute the postive fraction
pos_fraction = npos / nRows
neg_fraction = 1 - pos_fraction
pos_fraction, neg_fraction

(npos, pos_fraction), (nneg, neg_fraction)
```

[4]: ((9891, 0.20142551674982181), (39214, 0.7985744832501782))

[5]:
```
""" PROVIDED
Make some adjustments to the data.

For wind speed, NaNs are current represented by -999.
We will replace these with NaN.

For Latitude and Longitude, these are strings such as
28.0W. We will replace these with numerical values where
positive directions are N and E, and negative directions
are S and W.
"""
# Convert -999 values to NaNs. These are missing values
NaNvalue = -999
cyclones_nans = cyclones_full.replace(NaNvalue, np.nan).copy()

# Set the datatype of the categorical attributes
cate_attribs = ['Event', 'Status']
cyclones_nans[cate_attribs] = cyclones_full[cate_attribs].astype('category')

# Set the datatype of the Data attribute to datetime64[ns]
cyclones_nans['Date'] = cyclones_nans['Date'].astype('datetime64[ns]')

# Convert Latitude and Longitude into numerical values
def to_numerical(coord):
```

```python
    direction = re.findall(r'[NSWE]' , coord)[0]
    num = re.match('[\d]{1,3}.[\d]{0,1}' , coord)[0]

    # North and East are positive directions
    if direction in ['N', 'E']:
        return np.float(num)
    return -1. * np.float(num)

cyclones_nans['Latitude'] = cyclones_nans['Latitude'].apply(to_numerical)
cyclones_nans['Longitude'] = cyclones_nans['Longitude'].apply(to_numerical)
cyclones_nans[['Latitude', 'Longitude']].head(3)
```

[5]:

	Latitude	Longitude
0	28.0	-94.8
1	28.0	-95.4
2	28.0	-96.0

[6]:

```python
""" PROVIDED
Display the quantitiy of NaNs for each feature
"""
cyclones_nans.isna().sum()
```

[6]:

```
ID                      0
Name                    0
Date                    0
Time                    0
Event                   0
Status                  0
Latitude                0
Longitude               0
Maximum Wind            0
Minimum Pressure    30669
Low Wind NE         43184
Low Wind SE         43184
Low Wind SW         43184
Low Wind NW         43184
Moderate Wind NE    43184
Moderate Wind SE    43184
Moderate Wind SW    43184
Moderate Wind NW    43184
High Wind NE        43184
High Wind SE        43184
High Wind SW        43184
High Wind NW        43184
isTD                    0
dtype: int64
```

[7]:
```
""" PROVIDED
Display summary statistics for each feature of the dataframe
"""
cyclones_nans.describe()
```

[7]:
```
               Time      Latitude     Longitude  Maximum Wind  \
count  49105.000000  49105.000000  49105.000000  49105.000000
mean     910.125975     27.044904    -65.682533     52.005091
std      671.043363     10.077880     19.687240     27.681902
min        0.000000      7.200000   -359.100000    -99.000000
25%      600.000000     19.100000    -81.000000     35.000000
50%     1200.000000     26.400000    -68.000000     45.000000
75%     1800.000000     33.100000    -52.500000     70.000000
max     2330.000000     81.000000     63.000000    165.000000

       Minimum Pressure  Low Wind NE  Low Wind SE  Low Wind SW  Low Wind NW  \
count      18436.000000  5921.000000  5921.000000  5921.000000  5921.000000
mean         992.244250    81.865394    76.518325    48.647188    59.156393
std           19.113748    88.097930    87.563153    75.209183    77.568911
min          882.000000     0.000000     0.000000     0.000000     0.000000
25%          984.000000     0.000000     0.000000     0.000000     0.000000
50%          999.000000    60.000000    60.000000     0.000000    40.000000
75%         1006.000000   130.000000   120.000000    75.000000    90.000000
max         1024.000000   710.000000   600.000000   640.000000   530.000000

       Moderate Wind NE  Moderate Wind SE  Moderate Wind SW  Moderate Wind NW  \
count       5921.000000       5921.000000       5921.000000       5921.000000
mean          24.641952         23.029894         15.427293         18.403141
std           41.592337         42.017821         32.105372         35.411258
min            0.000000          0.000000          0.000000          0.000000
25%            0.000000          0.000000          0.000000          0.000000
50%            0.000000          0.000000          0.000000          0.000000
75%           40.000000         35.000000         20.000000         30.000000
max          360.000000        300.000000        330.000000        360.000000

       High Wind NE  High Wind SE  High Wind SW  High Wind NW
count   5921.000000   5921.000000   5921.000000   5921.000000
mean       8.110117      7.357710      5.130890      6.269211
std       19.792002     18.730334     14.033464     16.876623
min        0.000000      0.000000      0.000000      0.000000
25%        0.000000      0.000000      0.000000      0.000000
50%        0.000000      0.000000      0.000000      0.000000
75%        0.000000      0.000000      0.000000      0.000000
max      180.000000    250.000000    150.000000    180.000000
```

3 PRE-PROCESS DATA

[8]:
```
cyclones_nans.columns
```

[8]:
```
Index(['ID', 'Name', 'Date', 'Time', 'Event', 'Status', 'Latitude',
       'Longitude', 'Maximum Wind', 'Minimum Pressure', 'Low Wind NE',
       'Low Wind SE', 'Low Wind SW', 'Low Wind NW', 'Moderate Wind NE',
       'Moderate Wind SE', 'Moderate Wind SW', 'Moderate Wind NW',
       'High Wind NE', 'High Wind SE', 'High Wind SW', 'High Wind NW', 'isTD'],
      dtype='object')
```

[9]:
```
""" PROVIDED
Construct preprocessing pipeline
"""
dropped_features = ['ID', 'Name', 'Date', 'Time', 'Status', 'Event']
#selected_features = cyclones_nans.columns.drop(dropped_features)
selected_features = ['Latitude', 'Longitude', 'Low Wind SW', 'Moderate Wind NE',
                     'Moderate Wind SE', 'High Wind NW', 'isTD']

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

[10]:
```
""" PROVIDED
Pre-process the data using the defined pipeline
"""
processed_data = pipe.fit_transform(cyclones_nans)
nsamples, ncols = processed_data.shape
nsamples, ncols
```

[10]:
```
(5921, 7)
```

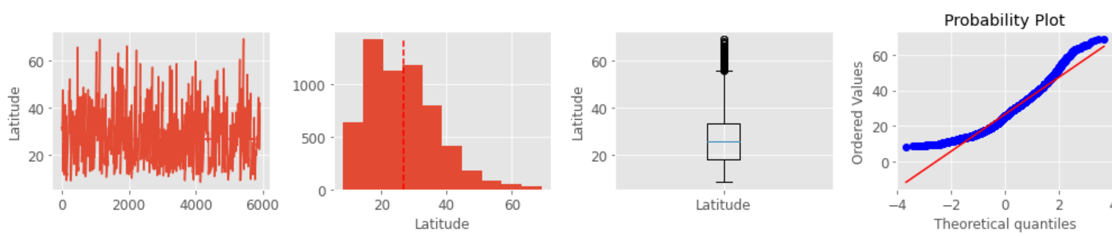
[11]:
```
""" PROVIDED
Verify all NaNs removed
"""
processed_data.isna().any()
```

[11]:
```
Latitude            False
Longitude           False
Low Wind SW         False
Moderate Wind NE    False
Moderate Wind SE    False
High Wind NW        False
isTD                False
dtype: bool
```

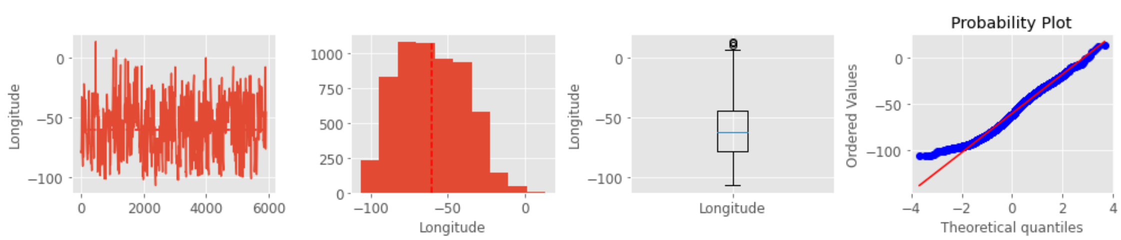
4 VISUALIZE DATA

[12]:
```python
""" PROVIDED
Display the distributions of the data
use visualize.featureplots
to generate trace plots, histograms, boxplots, and probability plots for
each feature.

A probability plot is utilized to evaulate the normality of a distribution.
The data are plot against a theoritical distribution, such that if the data
are normal, they'll follow the diagonal line. See the reference above for
more information.
"""
cdata = processed_data.astype('float64').copy()
visualize.featureplots(cdata.values, cdata.columns)
# You can right click to enable scrolling
```

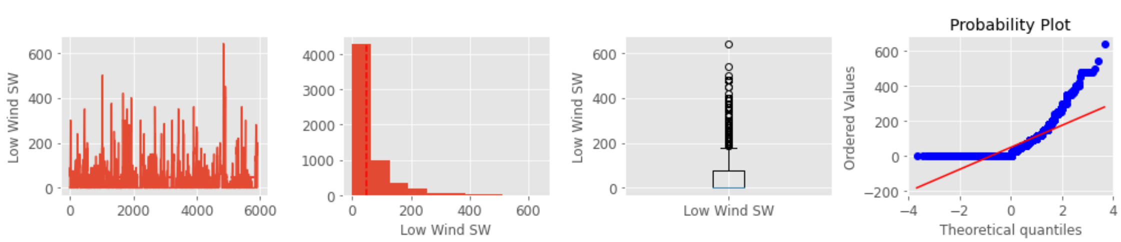
FEATURE: Latitude



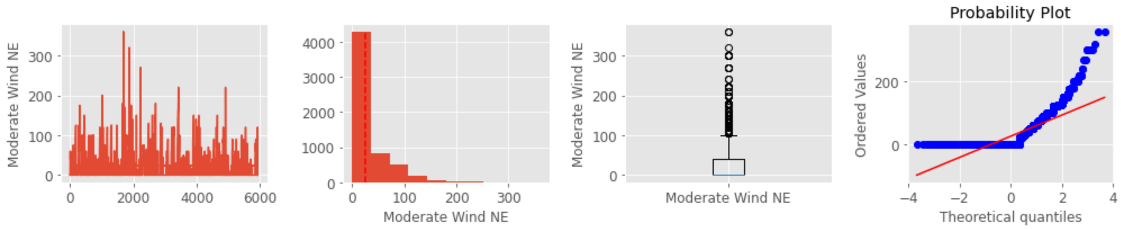
FEATURE: Longitude



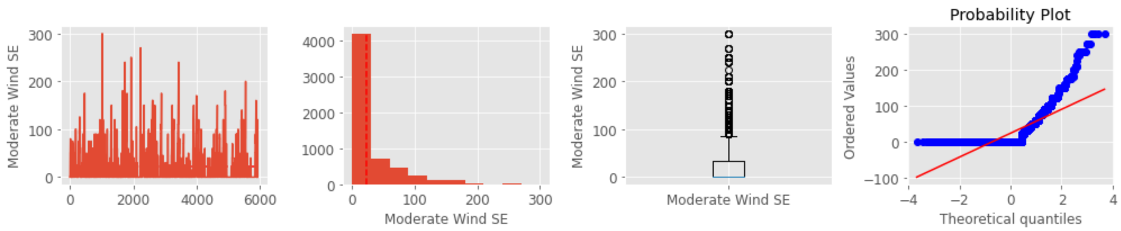
FEATURE: Low Wind SW



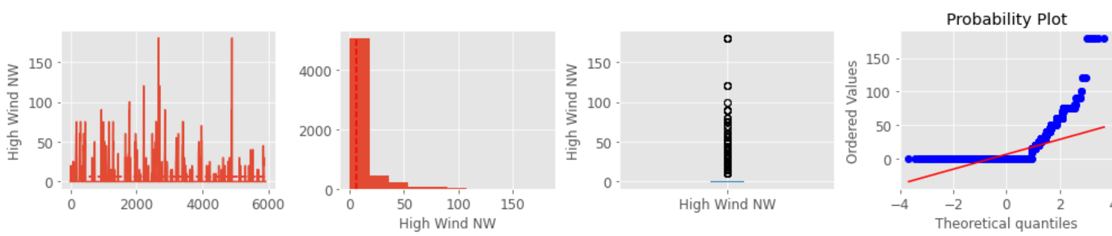
FEATURE: Moderate Wind NE



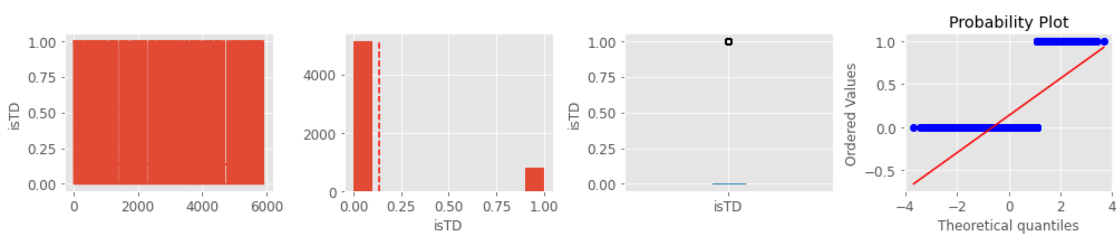
FEATURE: Moderate Wind SE



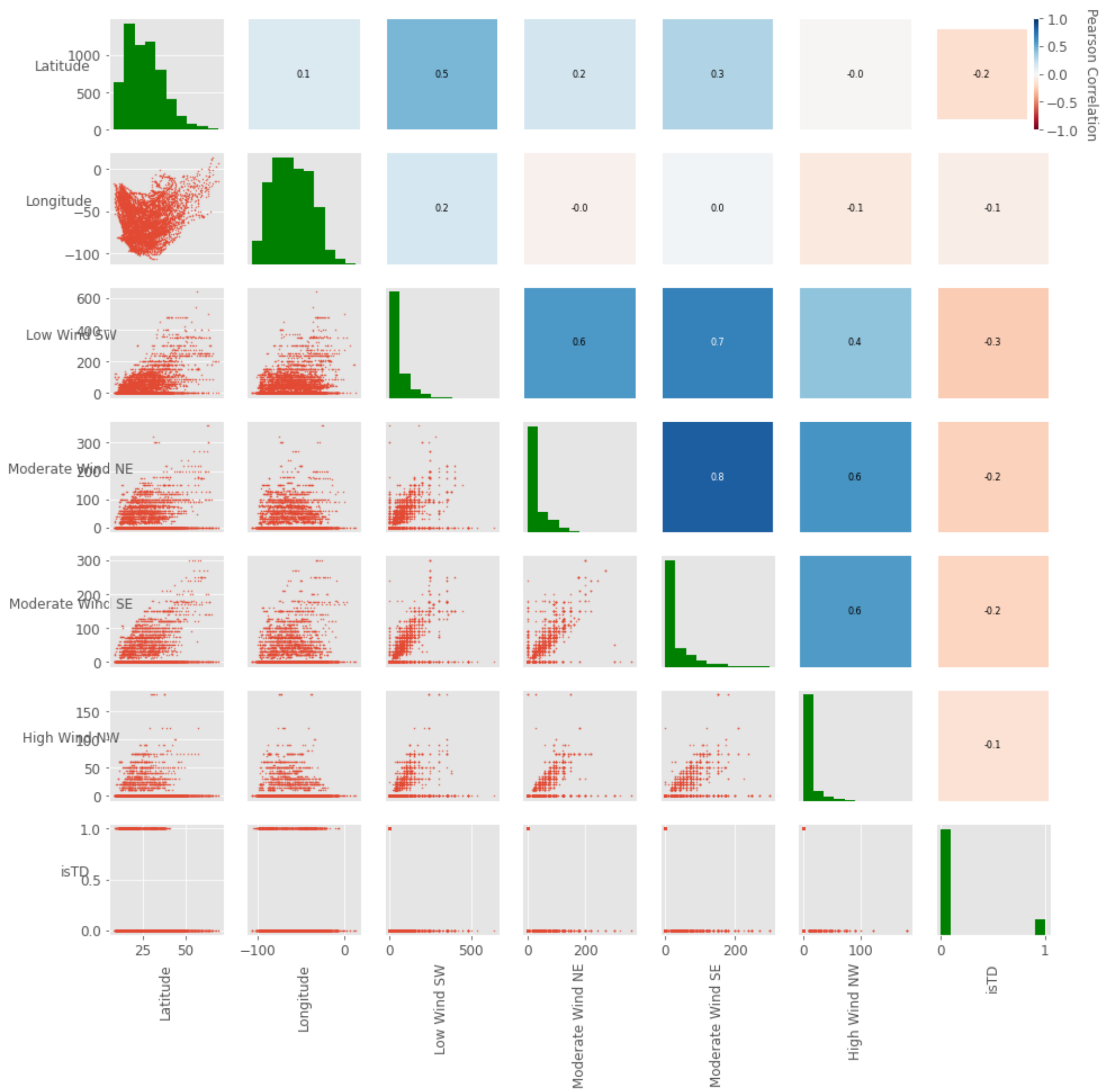
FEATURE: High Wind NW



FEATURE: isTD



[13]:
```
""" PROVIDED
Display the Pearson correlation between all pairs of the features
use visualize.scatter_corrplots
"""
visualize.scatter_corrplots(cdata.values, cdata.columns, corrfmt="%.1f",␣
↪FIGW=15)
```



[14]:
```
""" PROVIDED
Extract the positive and negative cases
"""
# Get the positions of the positive and negative labeled examples
pos_inds = processed_data['isTD'] == 1
neg_inds = processed_data['isTD'] == 0
```

```python
# Get the actual corresponding examples
pos = processed_data[pos_inds]
neg = processed_data[neg_inds]

# Positive Fraction
npos = pos_inds.sum()
nneg = nsamples - npos
pos_frac = npos / nsamples
neg_frac = 1 - pos_frac
(npos, pos_frac), (nneg, neg_frac)
```

[14]:
```
((788, 0.13308562742779936), (5133, 0.8669143725722006))
```

5 CLASSIFICATION

[15]:
```python
""" PROVIDED
Functions for exporting trees to .dot and .pngs
"""
from PIL import Image
def image_combine(ntrees, big_name='big_tree.png', fname_fmt='tree_%02d.png'):
    '''
    Function for combining some of the trees in the forest into on image
    Amalgamate the pngs of the trees into one big image
    PARAMS:
        ntrees: number of trees from the ensemble to export
        big_name: file name for the png containing all ntrees
        fname_fmt: file name format string used to read the exported files
    '''
    # Read the pngs
    imgs = [Image.open(fname_fmt % x) for x in range(ntrees)]

    # Determine the individual and total sizes
    widths, heights = zip(*(i.size for i in imgs))
    total_width = sum(widths)
    max_height = max(heights)

    # Create the combined image
    big_img = Image.new('RGB', (total_width, max_height))
    x_offset = 0
    for im in imgs:
        big_img.paste(im, (x_offset, 0))
        x_offset += im.size[0]
    big_img.save(big_name)
    print("Created %s" % big_name)
    return big_img
```

```python
def export_trees(forest, ntrees=3, fname_fmt='tree_%02d'):
    '''
    Write trees into inidividual files from the forest
    PARAMS:
        forest: ensemble of trees classifier
        ntrees: number of trees from the ensemble to export
        fname_fmt: file name format string used to name the exported files
    '''
    for t in range(ntrees):
        estimator = forest.estimators_[t]
        basename = fname_fmt % t
        fname = basename + '.dot'
        pngname = basename + '.png'
        export_graphviz(estimator, out_file=fname, rounded=True, filled=True)
        # Command line instruction to execute dot and create the image
        !dot -Tpng {fname} > {pngname}
        print("Created %s and %s" % (fname, pngname))
```

[42]:
```python
""" TODO
Split the data into X (i.e. the inputs) and y (i.e. the outputs).
Recall we are predicting isTD.

Hold out a subset of the data, before training and cross validation
using train_test_split, with stratification, and a test_size
fraction of .2. See the sklearn API for more details

For this exploratory section, the held out set of data is a validation set.
"""
# TODO: Separate X and y. We are predicting isTD
X = processed_data[processed_data.columns.drop(['isTD'])]
y = processed_data['isTD']

# TODO: Hold out 20% of the data for validation
Xtrain, Xval, ytrain, yval = train_test_split(X, y, test_size=0.2, stratify=y)
```

```
       Latitude  Longitude  Low Wind SW  Moderate Wind NE  Moderate Wind SE  \
43104      30.3      -78.3          0.0               0.0               0.0
43105      31.0      -78.8          0.0               0.0               0.0
43106      31.5      -79.0          0.0               0.0               0.0
43107      31.6      -79.1          0.0               0.0               0.0
43108      31.6      -79.2         50.0               0.0               0.0
…           …          …          …                 …                 …
49100      41.3      -50.4        180.0             120.0             120.0
49101      41.9      -49.9        180.0             120.0             120.0
49102      41.5      -49.2        200.0             120.0             120.0
49103      40.8      -47.5        180.0               0.0               0.0
```

```
49104      40.7      -45.4       150.0             0.0              0.0

       High Wind NW
43104           0.0
43105           0.0
43106           0.0
43107           0.0
43108           0.0
…             …
49100           0.0
49101           0.0
49102           0.0
49103           0.0
49104           0.0

[5921 rows x 6 columns]
```

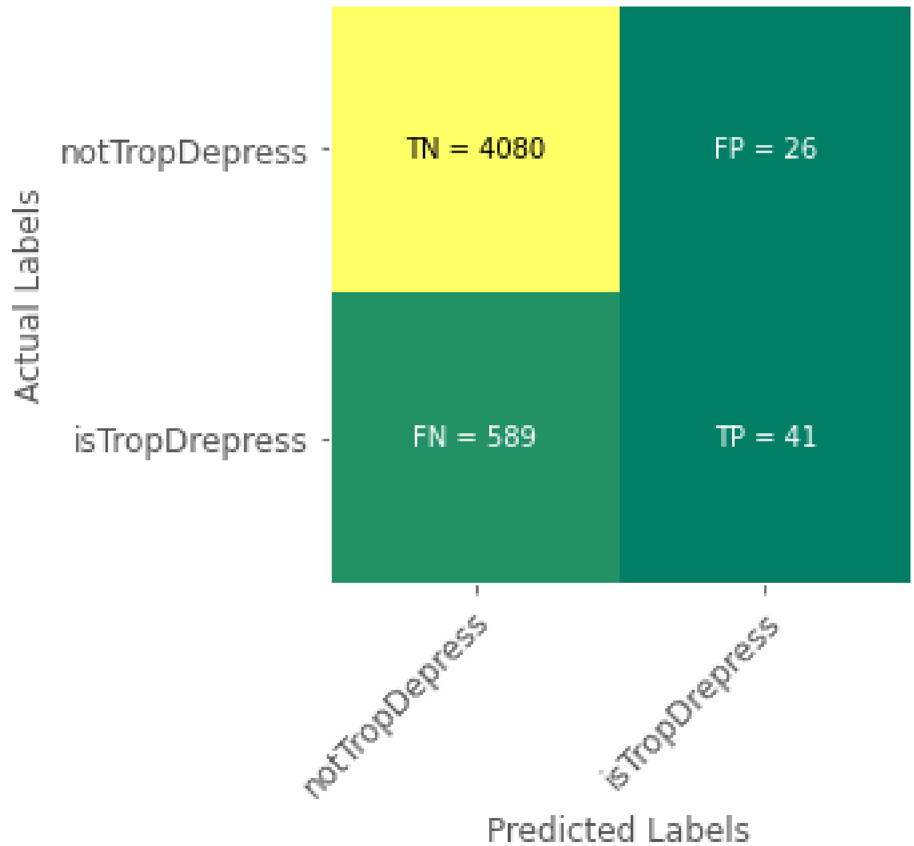
6 DECISION TREE CLASSIFIER

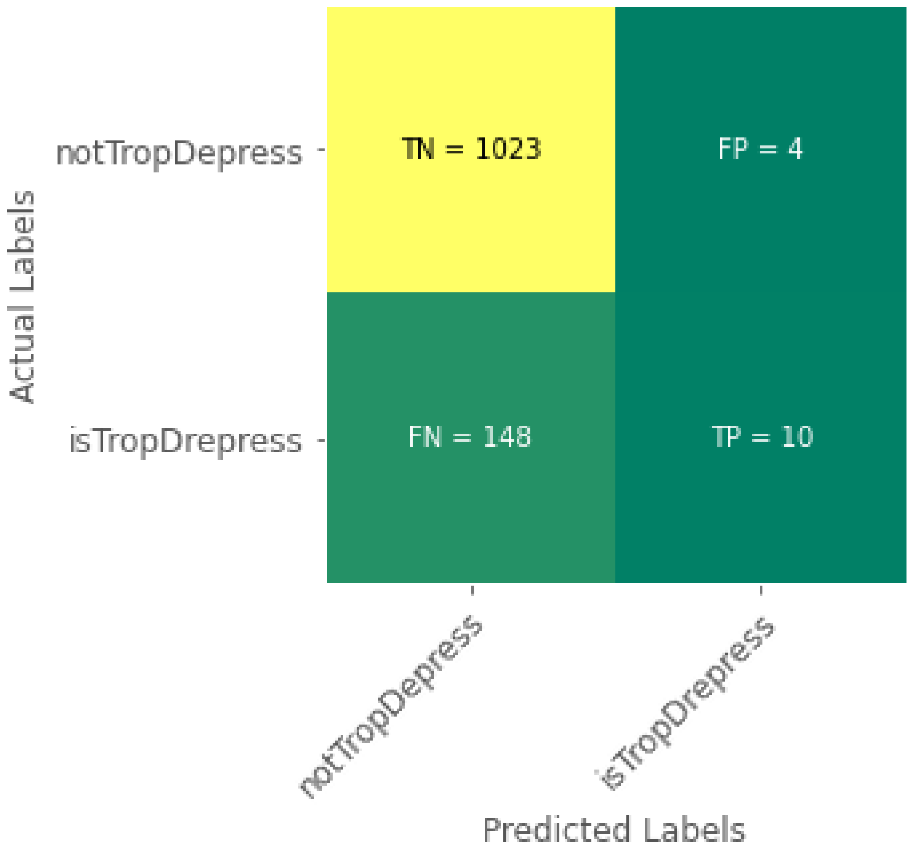
[17]:
```
""" PROVIDED
Create and train DecisionTree for comparision with the ensemble methods
"""
tree_clf = DecisionTreeClassifier(max_depth=200, max_leaf_nodes=10)
tree_clf.fit(Xtrain, ytrain)
```

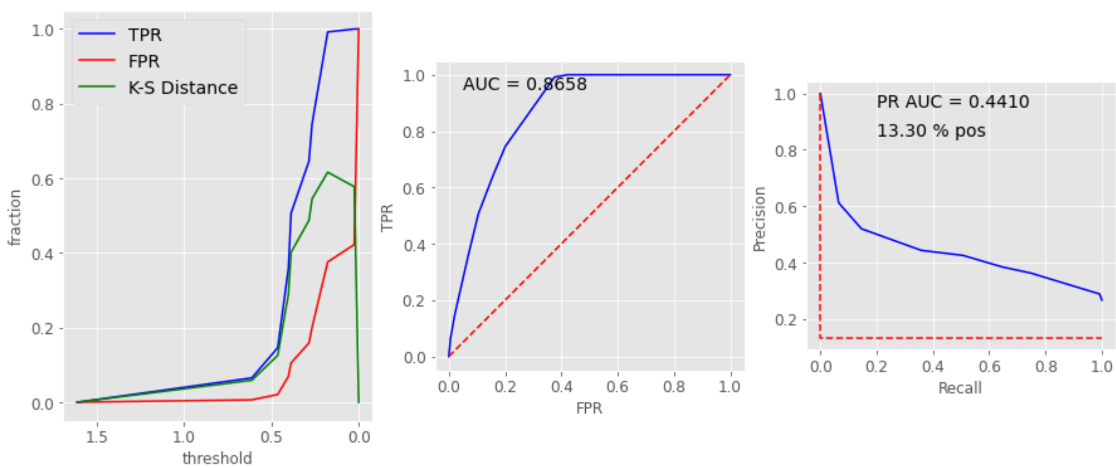
[17]:
```
DecisionTreeClassifier(max_depth=200, max_leaf_nodes=10)
```

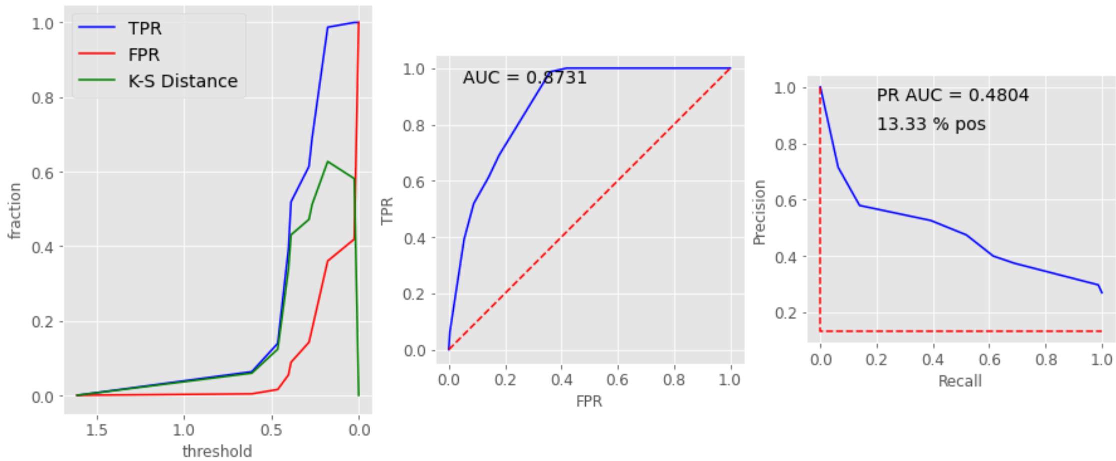
[18]:
```
""" PROVIDED
Compute the predictions, prediction probabilities, and the accuracy scores
for the trianing and validation sets
"""
# Compute the model's predictions
dt_preds = tree_clf.predict(Xtrain)
dt_preds_val = tree_clf.predict(Xval)

# Compute the prediction probabilities
dt_proba = tree_clf.predict_proba(Xtrain)
dt_proba_val = tree_clf.predict_proba(Xval)

# Compute the model's mean accuracy
dt_score = tree_clf.score(Xtrain, ytrain)
dt_score_val = tree_clf.score(Xval, yval)

# Confusion Matrix
dt_cmtx = confusion_matrix(ytrain, dt_preds)
dt_cmtx_val = confusion_matrix(yval, dt_preds_val)
metrics_plots.confusion_mtx_colormap(dt_cmtx, targetnames, targetnames)
```

```
metrics_plots.confusion_mtx_colormap(dt_cmtx_val, targetnames, targetnames)

# KS, ROC, and PRC Curves
dt_roc_prc_results = metrics_plots.ks_roc_prc_plot(ytrain, dt_proba[:,1])
dt_roc_prc_results_val = metrics_plots.ks_roc_prc_plot(yval, dt_proba_val[:,1])
```



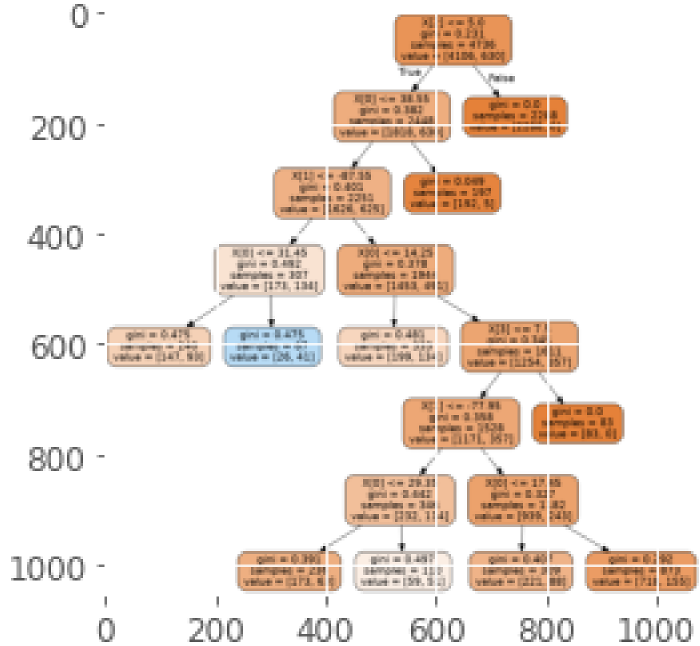

TN = 1023
FP = 4
FN = 148
TP = 10
notTropDepress
isTropDrepress
Actual Labels
notTropDepress
isTropDrepress
Predicted Labels


TPR
FPR
K-S Distance
fraction
threshold
AUC = 0.8658
TPR
FPR
PR AUC = 0.4410
13.30 % pos
Precision
Recall



[19]:
```python
""" PROVIDED
Export the tree as a .dot file and create the png
"""
fname = 'tree.dot'
pngname = 'tree.png'
export_graphviz(tree_clf, out_file=fname, rounded=True, filled=True)

# If the following command does not work, you can manually convert
# the dot file into a png here: https://onlineconvertfree.com/convert-format/
↪dot-to-png/
!dot -Tpng {fname} > {pngname}
```

[20]:
```python
'''
PROVIDED
Display the tree file
'''
img = mpimg.imread('tree.png')
plt.imshow(img)
plt.show()
```



7 RANDOM FOREST CLASSIFIER

[21]:
```python
""" TODO
Create and train RandomForests
Explore various configurations of the hyper-parameters.
Train the models on the training set and evaluate them for the training and
validation sets.
Take a look at the API and the book for the meaning and impact of different
hyper-parameters
"""
forest_clf = RandomForestClassifier(random_state = 1,
                                    n_estimators = 600,
                                    max_depth = 10,
                                    min_samples_split = 5,  min_samples_leaf = 1)
forest_clf.fit(Xtrain, ytrain)
```

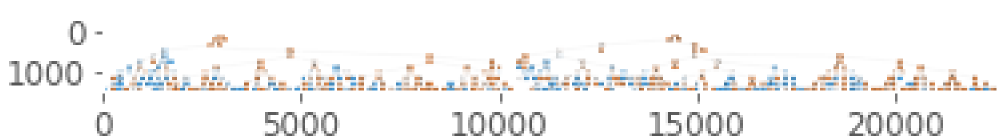
[21]:
```
RandomForestClassifier(max_depth=10, min_samples_split=5, n_estimators=600,
                       random_state=1)
```

[22]:
```python
""" PROVIDED
Export some trees from your favorite model as a .dot file
We can use the estimators_ attribute of the forest to get a list of the trees

Amalgamate the pngs of the trees into one big image
```

```
"""
ntrees = 2

'''
If the dot command does not work on your computer, please modify the␣
 ↪export_trees
function by commenting out the line where the dot command is being invoked
Then you can manually convert each dot file into a png file at the following␣
 ↪website
https://onlineconvertfree.com/convert-format/dot-to-png/
After converting all of the dot files into a png, you should be able to use the
image_comibne() function
'''
export_trees(forest_clf, ntrees, fname_fmt='e_rf_model_%02d')
big_img = image_combine(ntrees, big_name='e_rf_model.png',
                        fname_fmt='e_rf_model_%02d.png')
```

```
Created e_rf_model_00.dot and e_rf_model_00.png
Created e_rf_model_01.dot and e_rf_model_01.png
Created e_rf_model.png
```

[23]:
```
'''
PROVIDED
Display the tree file
'''
img = mpimg.imread('e_rf_model.png')
plt.imshow(img)
plt.show()
```



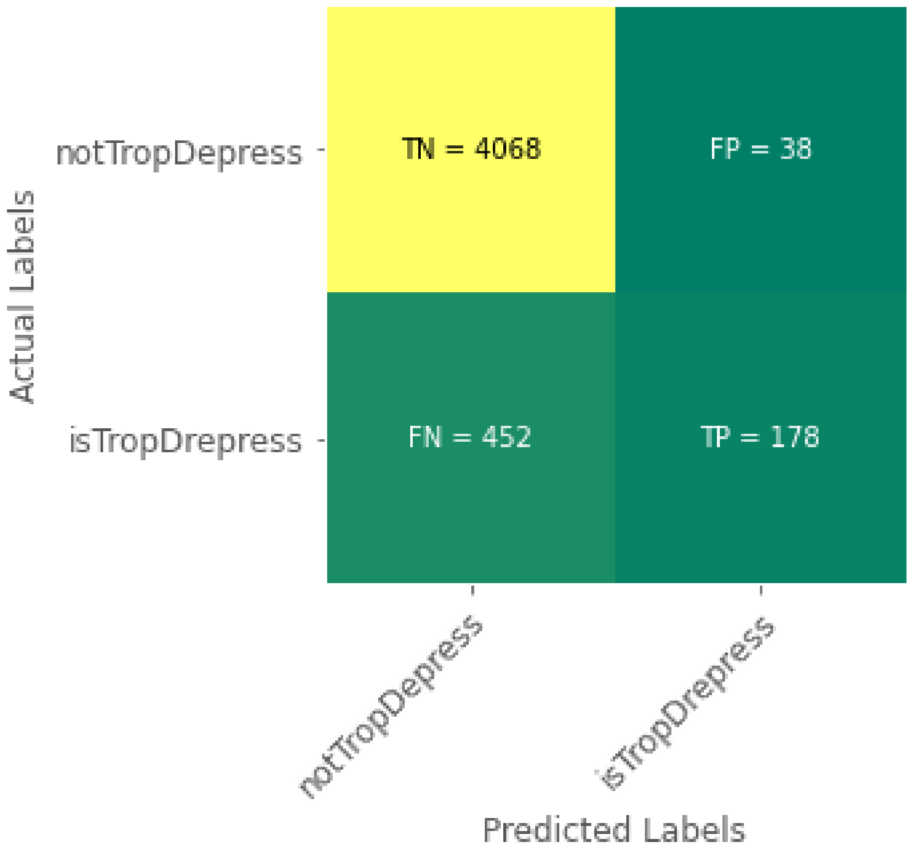
7.0.1 TRAINING AND VALIDATION RESULTS

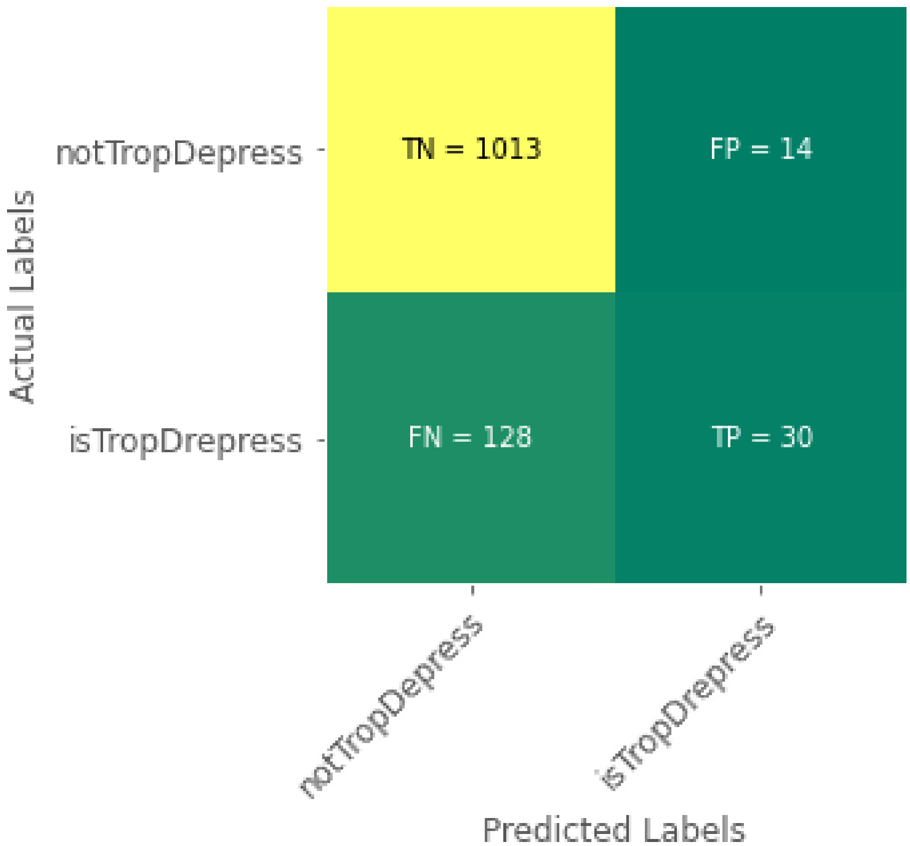
[27]:
```
""" TODO
Compute the predictions, prediction probabilities, and the accuracy scores
for the training and validation sets for the learned instance of the model
"""
# TODO: Compute the model's predictions. use model.predict()
forest_preds = forest_clf.predict(Xtrain)
forest_preds_val = forest_clf.predict(Xval)
```

```
# TODO: Compute the prediction probabilities. use model.predict_proba()
forest_proba = forest_clf.predict_proba(Xtrain)
forest_proba_val = forest_clf.predict_proba(Xval)


# TODO: Compute the model's mean accuracy. use model.score()
forest_score = forest_clf.score(Xtrain, ytrain)
forest_score_val = forest_clf.score(Xval,yval)
```

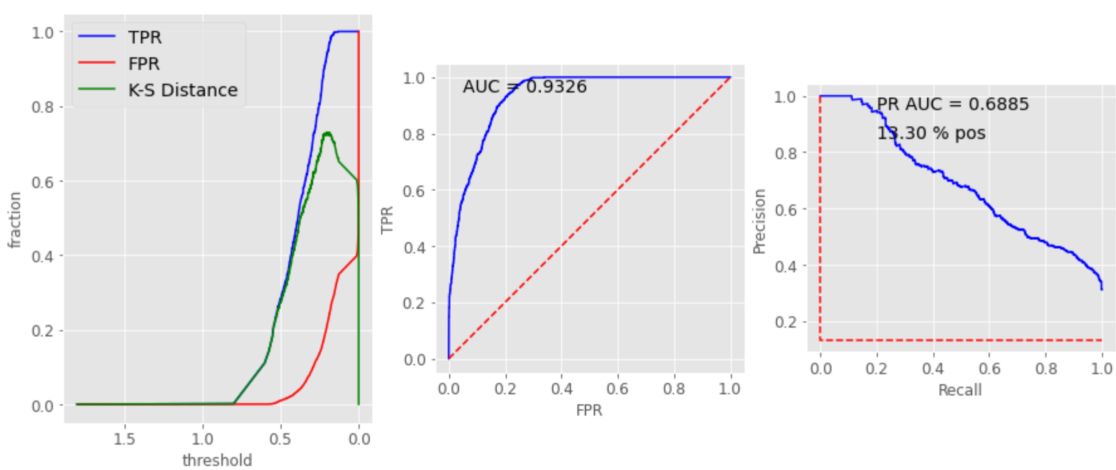
[28]:

```
""" TODO
Display the confusion matrix, KS plot, ROC curve, and PR curve for the training
and validation sets using metrics_plots.ks_roc_prc_plot

The red dashed line in the ROC and PR plots are indicative of the expected
performance for a random classifier, which would predict postives at the
rate of occurance within the data set
"""
# TODO: Confusion Matrix
forest_cmtx = confusion_matrix(ytrain, forest_preds)
forest_cmtx_val = confusion_matrix(yval, forest_preds_val)
metrics_plots.confusion_mtx_colormap(forest_cmtx, targetnames, targetnames)
metrics_plots.confusion_mtx_colormap(forest_cmtx_val, targetnames, targetnames)


# TODO: KS, ROC, and PRC Curves
forest_roc_prc_results = metrics_plots.ks_roc_prc_plot(ytrain, forest_proba[:
 ↪,1])
forest_roc_prc_results_val = metrics_plots.ks_roc_prc_plot(yval,␣
 ↪forest_proba_val[:,1])
```

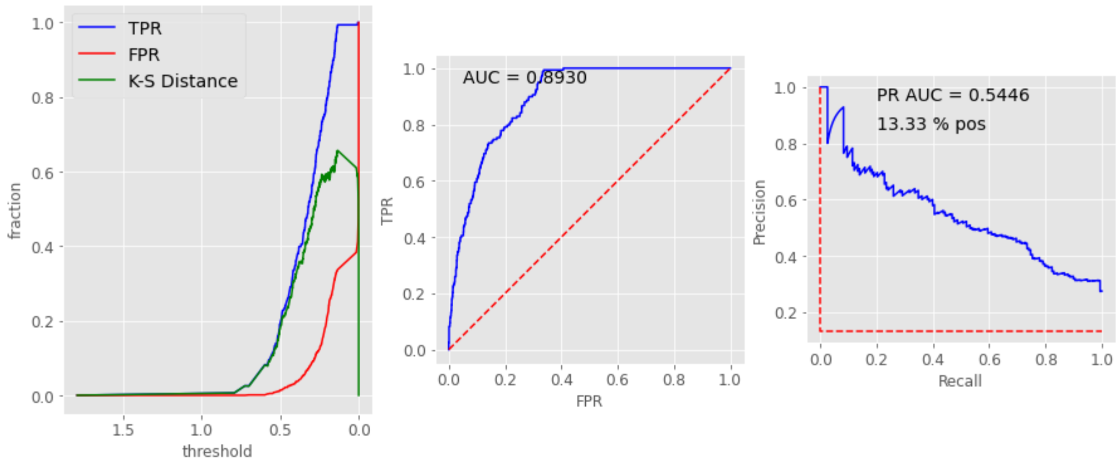

TN = 4068
FP = 38
FN = 452
TP = 178
notTropDepress
isTropDrepress
Actual Labels
notTropDepress
isTropDrepress
Predicted Labels



notTropDepress
isTropDrepress
Actual Labels
TN = 1013
FP = 14
FN = 128
TP = 30
notTropDepress
isTropDrepress
Predicted Labels



TPR
FPR
K-S Distance
fraction
threshold
AUC = 0.9326
TPR
FPR
PR AUC = 0.6885
13.30 % pos
Precision
Recall



8 ADABOOSTING

[29]:
```
""" TODO
Create and train a Boosting model
Explore various boosting models to improve your validation performance
Train the models on the training set and evaluate them for the training and
validation sets. Try boosting the benmark tree_clf
"""
ABC = AdaBoostClassifier(base_estimator = tree_clf, n_estimators=100,␣
 ↪learning_rate=2, random_state=1)
ABC.fit(Xtrain,ytrain)
```

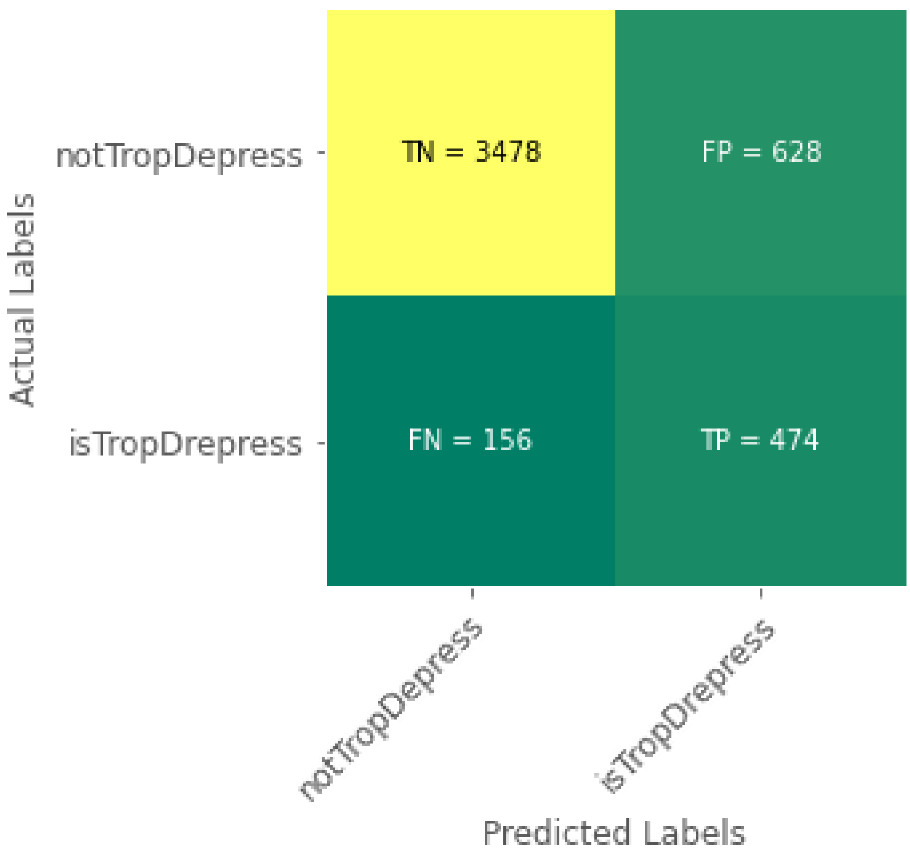
[29]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=200,
                                                         max_leaf_nodes=10),
                   learning_rate=2, n_estimators=100, random_state=1)
```

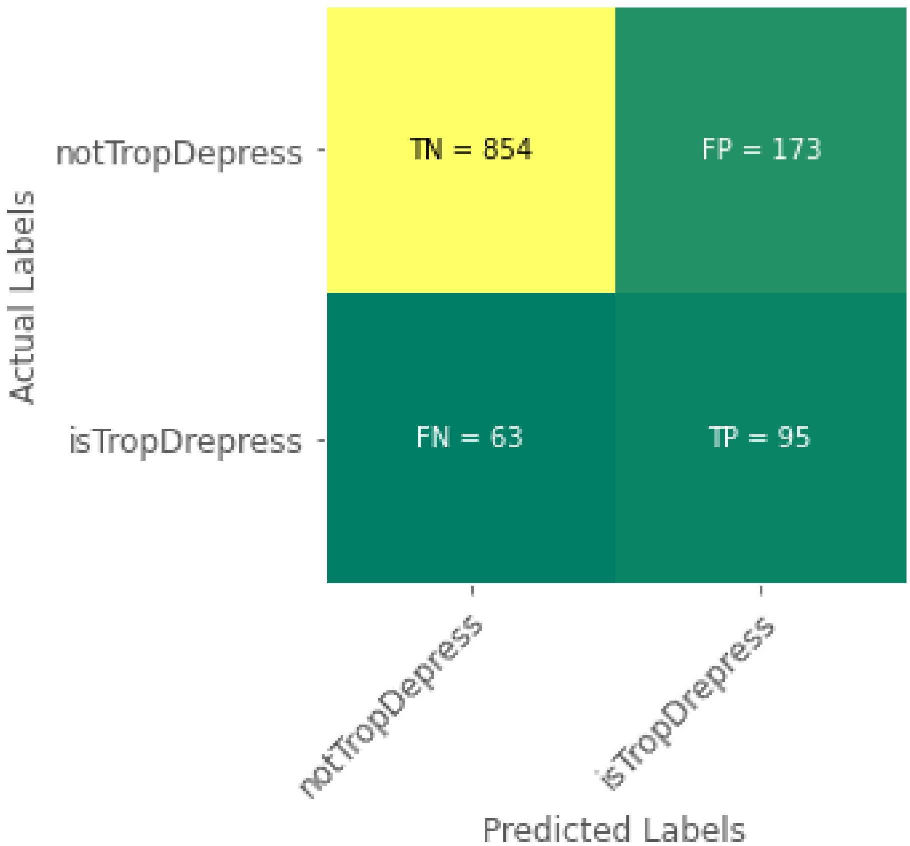
8.0.1 TRAINING AND VALIDATION RESULTS

[30]:
```
""" TODO
Compute the predictions, prediction probabilities, and the accuracy scores
for the trianing and validation sets
"""
# TODO: Compute the model's predictions
ABC_preds = ABC.predict(Xtrain)
ABC_preds_val = ABC.predict(Xval)


# TODO: Compute the prediction probabilities
ABC_proba = ABC.predict_proba(Xtrain)
ABC_proba_val = ABC.predict_proba(Xval)
```

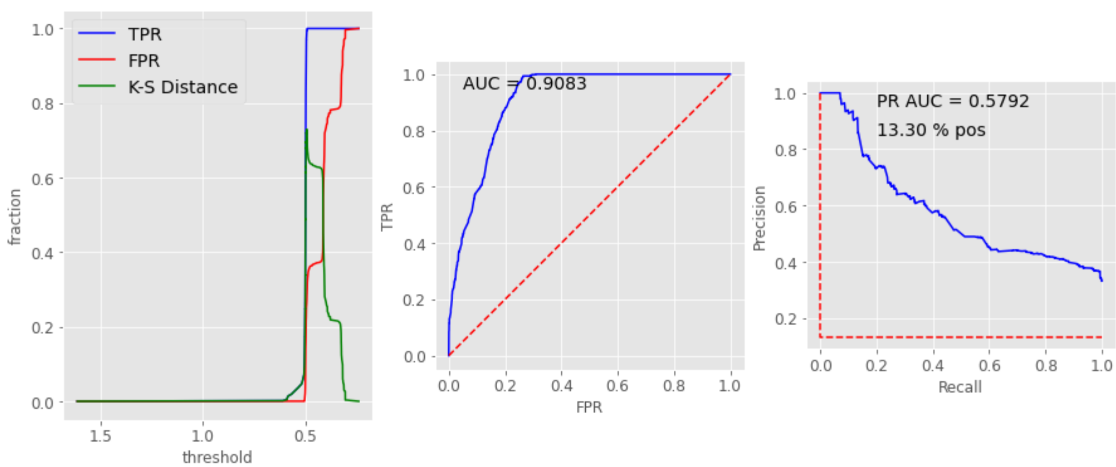
```python
# TODO: Compute the model's scores
ABC_score = ABC.score(Xtrain, ytrain)
ABC_score_val = ABC.score(Xval,yval)
```

[31]:
```python
""" TODO
Display the confusion matrix, KS plot, ROC curve, and PR curve for the
training and validation sets using metrics_plots.ks_roc_prc_plot
"""
# TODO: Confusion Matrix
ABC_cmtx = confusion_matrix(ytrain, ABC_preds)
ABC_cmtx_val = confusion_matrix(yval, ABC_preds_val)
metrics_plots.confusion_mtx_colormap(ABC_cmtx, targetnames, targetnames)
metrics_plots.confusion_mtx_colormap(ABC_cmtx_val, targetnames, targetnames)


# TODO: KS, ROC, and PRC Curves
ABC_roc_prc_results = metrics_plots.ks_roc_prc_plot(ytrain, ABC_proba[:,1])
ABC_roc_prc_results_val = metrics_plots.ks_roc_prc_plot(yval, ABC_proba_val[:
 ↪,1])
```

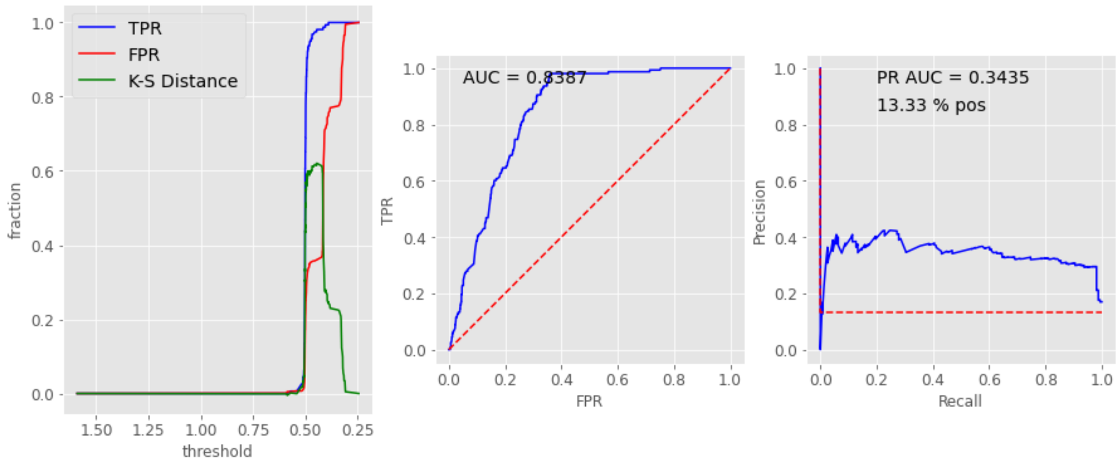




TN = 854
FP = 173
FN = 63
TP = 95
notTropDepress
isTropDrepress
Actual Labels
notTropDepress
isTropDrepress
Predicted Labels

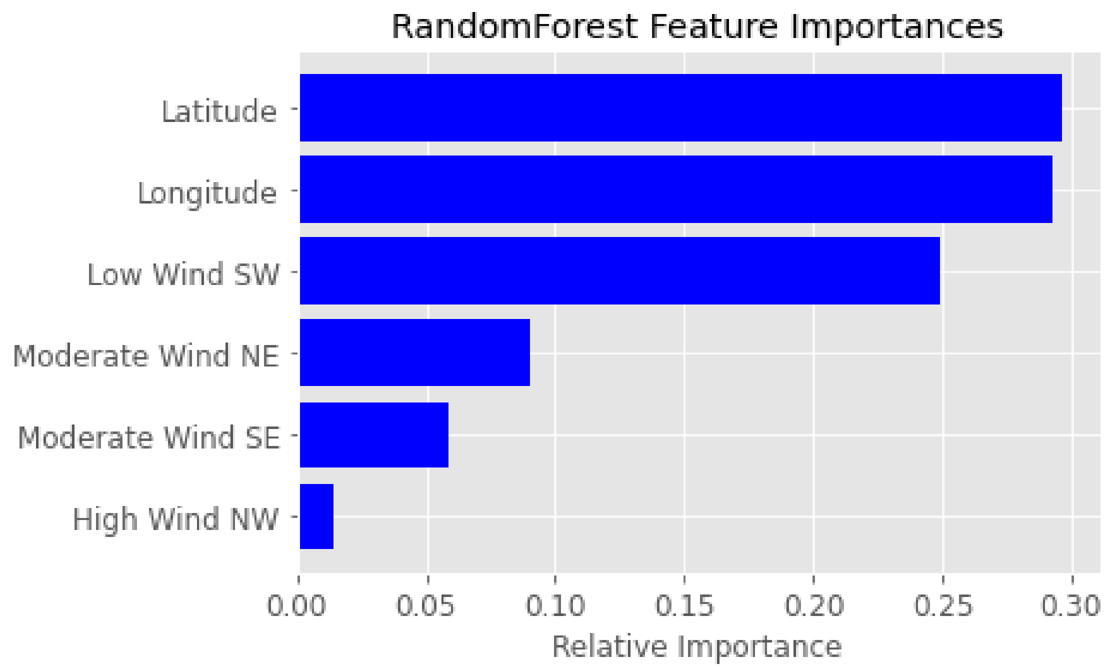


TPR
FPR
K-S Distance
fraction
threshold
AUC = 0.9083
TPR
FPR
PR AUC = 0.5792
13.30 % pos
Precision
Recall

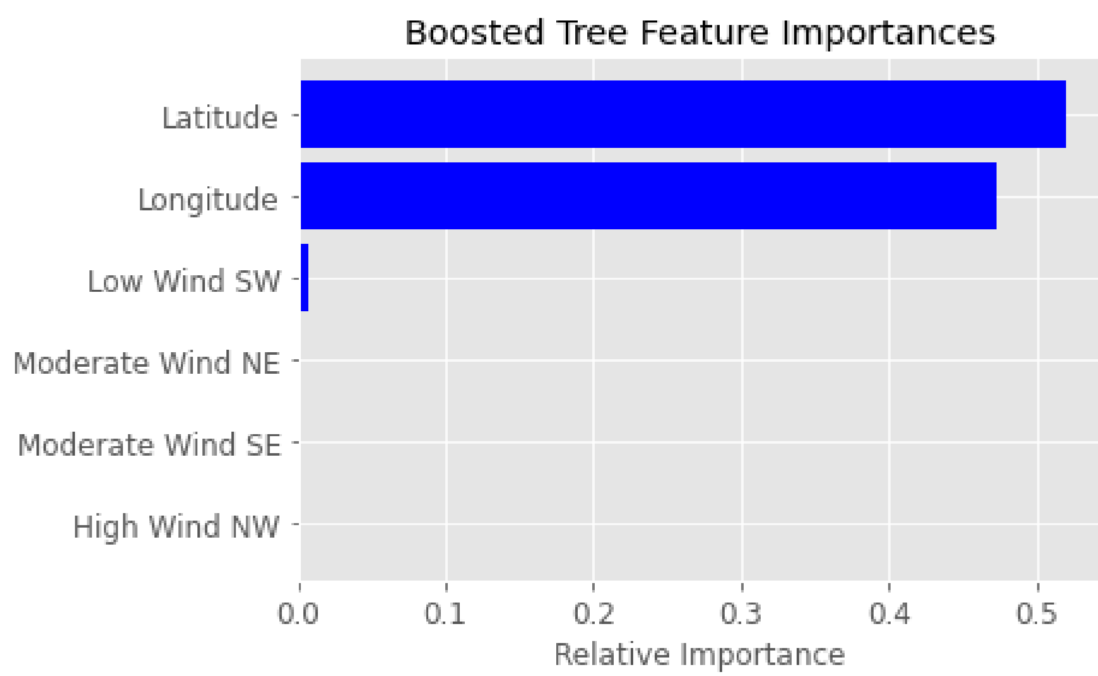


9 FEATURE IMPORTANCE

[51]:
```
""" TODO
Display the feature imporantances
see the API for RandomForests and boosted tree
you can create a DataFrame to help with the display
"""
forest_importances = forest_clf.feature_importances_
forest_indices = np.argsort(forest_importances)
plt.title('RandomForest Feature Importances')
plt.barh(range(len(forest_indices)), forest_importances[forest_indices],␣
 ↪color='b', align='center')
plt.yticks(range(len(forest_indices)), [selected_features[i] for i in␣
 ↪forest_indices])
plt.xlabel('Relative Importance')
plt.show()
print(forest_importances)

ABC_importances = ABC.feature_importances_
ABC_indices = np.argsort(ABC_importances)
plt.title('Boosted Tree Feature Importances')
plt.barh(range(len(ABC_indices)), ABC_importances[ABC_indices], color='b',␣
 ↪align='center')
plt.yticks(range(len(ABC_indices)), [selected_features[i] for i in ABC_indices])
plt.xlabel('Relative Importance')
plt.show()
print(ABC_importances)
```




```
[0.29597145 0.2925695  0.24900803 0.08998635 0.05837677 0.01408791]
```




```
[5.19335128e-01 4.73329236e-01 6.29562134e-03 8.38328613e-04
 2.01686406e-04 0.00000000e+00]
```

10 DISCUSSION

1. In a few paragraphs, discuss and interpret the report of your results for the RandomForestClassifier. Describe their meaning in terms of the context of predicting tropical depressions and the potential impact of various features. Talk about how you selected the hyper parameters. Describe how performance changes over the hyper-parameter space.

2. Describe the impact of boosting in 1 or 2 paragraphs

1. Going off of the previous homeworks, we know that the higher the AUC is, the more accurate the model is. The RFC that I used had an AUC of .9326, which is excellent. Also, based off the definition of what a tropical depression is from weather.gov, "A tropical depression is a tropical cyclone that has maximum sustained surface winds (one-minute average) of 38 mph (33 knots) or less." we know that location and wind speed is important. Comparing this to my RFC feature importances, it is correct in looking at latitude, longitude, and low wind speed for predicting if there is a tropical depression. As for picking parameters, I just played around with different values until I found an AUC that I was comfortable with. Since yyperparameters directly control the behaviour a training algorithm, they significantly impact the performance of the model is being trained. For example, a low learning rate will miss important patterns in the data, while hight learning rate may have collisions.

2. With boosting, we can take a model and its predictions and make it a base for making more accurate predictions. This is seen by looking at tree_clf AUC .8658 and PR AUC .4410, This is decent, but using tree_clf as a base_estimator, we can improve these scores to AUC .9083 and PR AUC .5792. Clearly, boosting helps create more accurate predictions vs non-boosting.

```
[ ]:
```