homework10

November 21, 2019

NAME: Jacob Duvall
SECTION: C S-5970-995
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. If you have any questions, please post them to the Canvas Discussion.

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 ml_practices 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

[214]:
```
import sys

# THESE 3 IMPORTS ARE CUSTOM .py FILES AND CAN BE FOUND
# ON THE SERVER AND GIT
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 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, GridSearchCV
from sklearn.metrics import explained_variance_score, confusion_matrix
from sklearn.metrics import f1_score, mean_squared_error, roc_curve, auc
from sklearn.svm import SVR
from sklearn.externals 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')
```

[215]:
```
"""
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()
```

[215]: PosixPath('/home/jovyan/homework/hww10')

2 LOAD DATA

[216]:
```
# TODO: set appropriately
filename = '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
```

[217]:
```
""" 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)
```

[217]:
```
((9891, 0.20142551674982181), (39214, 0.7985744832501782))
```

[218]:
```
""" 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):
    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)
```

[218]:

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

[219]:
```
""" PROVIDED
Display the quantitiy of NaNs for each feature
"""
cyclones_nans.isna().sum()
```

[219]:
```
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
```

[220]:
```
""" PROVIDED
Display summary statistics for each feature of the dataframe
"""
cyclones_nans.describe()
```

[220]:

	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

[221]:
```
cyclones_nans.columns
```

[221]:
```
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')
```

[222]:
```
""" 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())
])
```

[223]:
```
""" PROVIDED
Pre-process the data using the defined pipeline
"""
processed_data = pipe.fit_transform(cyclones_nans)
nsamples, ncols = processed_data.shape
nsamples, ncols
```

[223]:
```
(5921, 7)
```

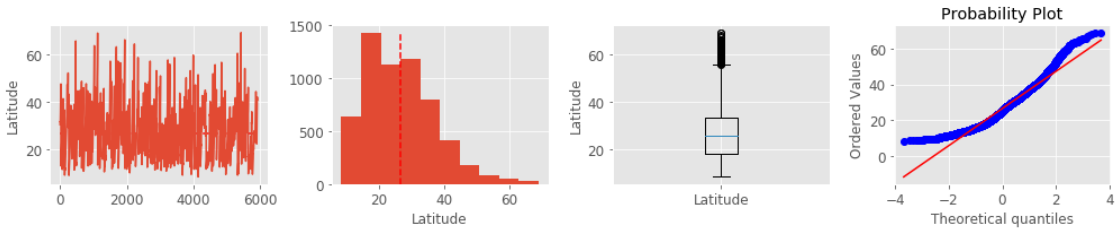
[224]:
```
""" PROVIDED
Verify all NaNs removed
"""
processed_data.isna().any()
```

[224]:
```
Latitude            False
Longitude           False
Low Wind SW         False
Moderate Wind NE    False
Moderate Wind SE    False
High Wind NW        False
isTD                False
dtype: bool
```

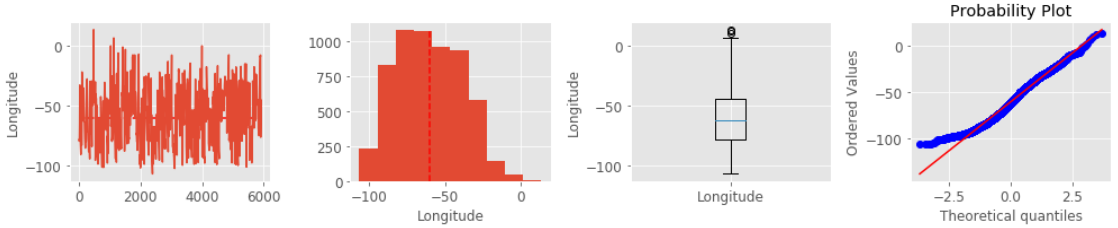
4 VISUALIZE DATA

[225]:
```
""" 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
```

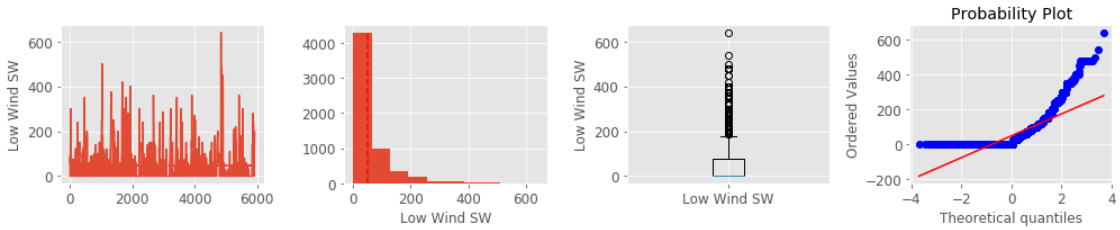
FEATURE: Latitude



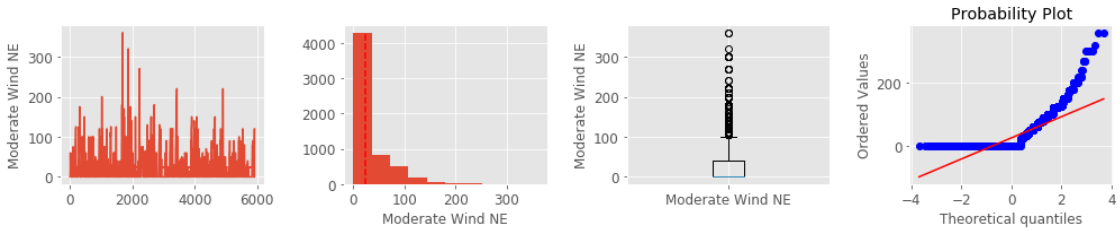
FEATURE: Longitude



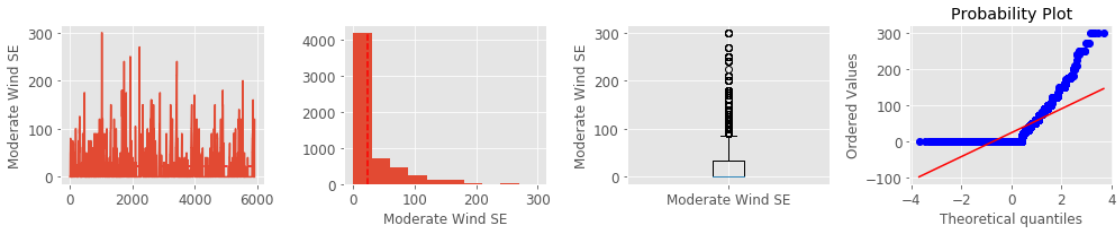
FEATURE: Low Wind SW



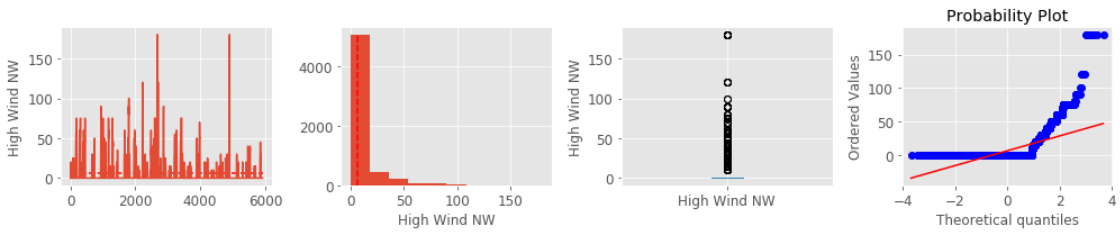
FEATURE: Moderate Wind NE



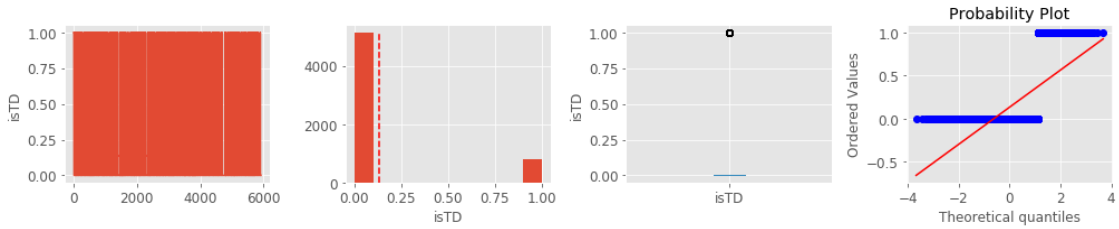
FEATURE: Moderate Wind SE



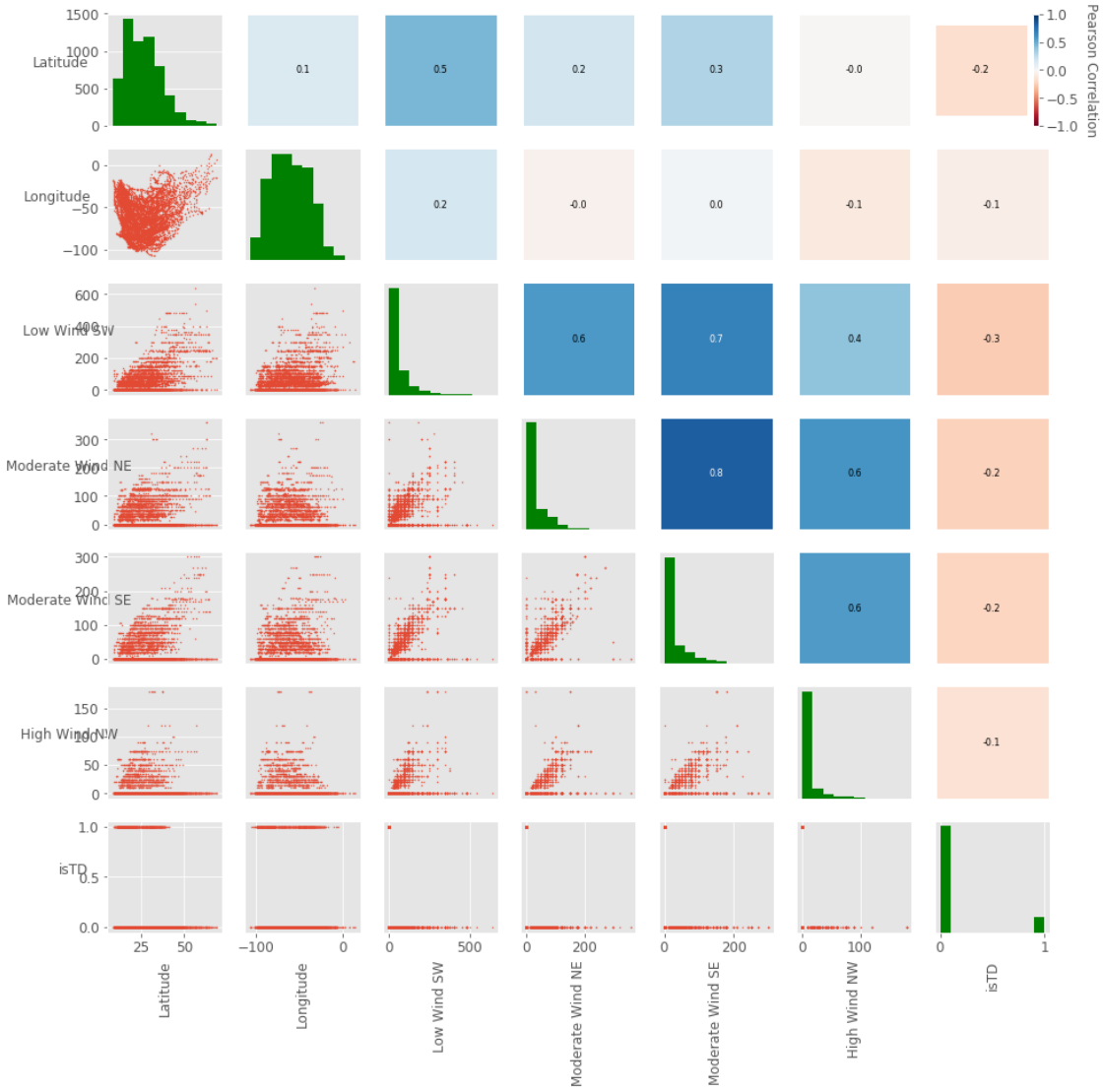
FEATURE: High Wind NW



FEATURE: isTD



[226]:
```
""" 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)
```



[227]:
```
""" 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

# 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)
```

[227]: ((788, 0.13308562742779936), (5133, 0.8669143725722006))

5 CLASSIFICATION

[228]:
```
""" 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

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))
```

[229]:
```
""" 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 = cdata.drop('isTD', axis=1)
y = cdata['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)
Xtrain.head()
```

[229]:

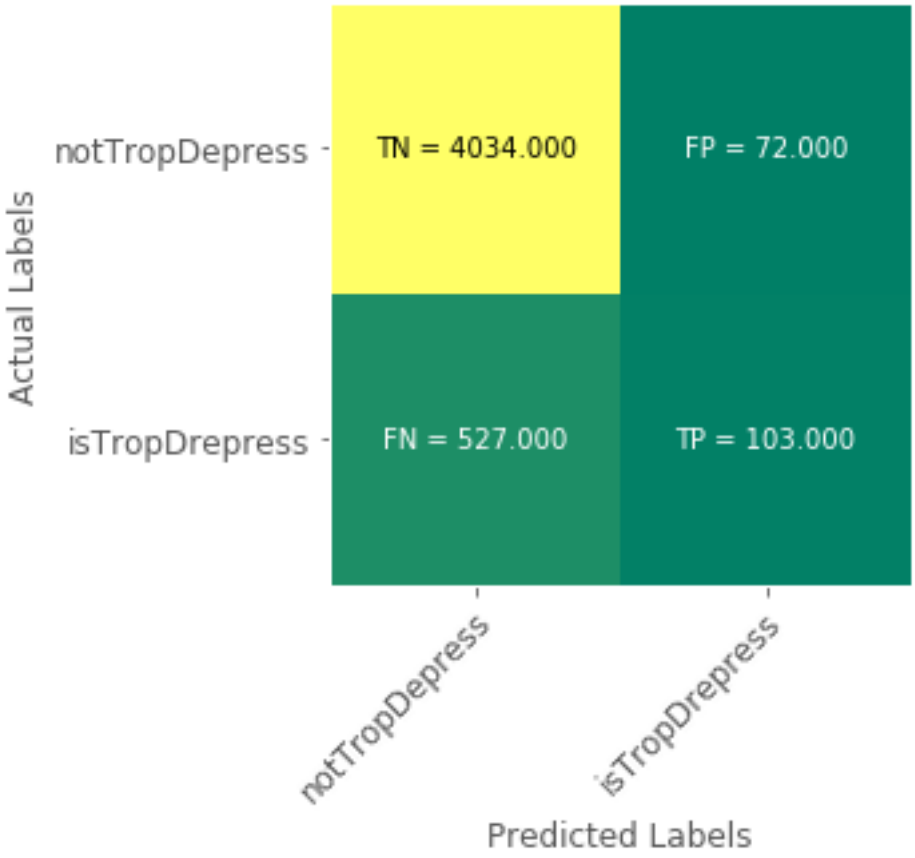
	Latitude	Longitude	Low Wind SW	Moderate Wind NE	Moderate Wind SE	High Wind NW
45159	35.6	-98.8	0.0	15.0	0.0	0.0
48962	10.7	-28.6	0.0	0.0	0.0	0.0
48184	16.9	-40.7	0.0	0.0	0.0	0.0
43161	25.7	-90.4	25.0	0.0	0.0	0.0
47775	48.1	-34.6	0.0	0.0	0.0	0.0

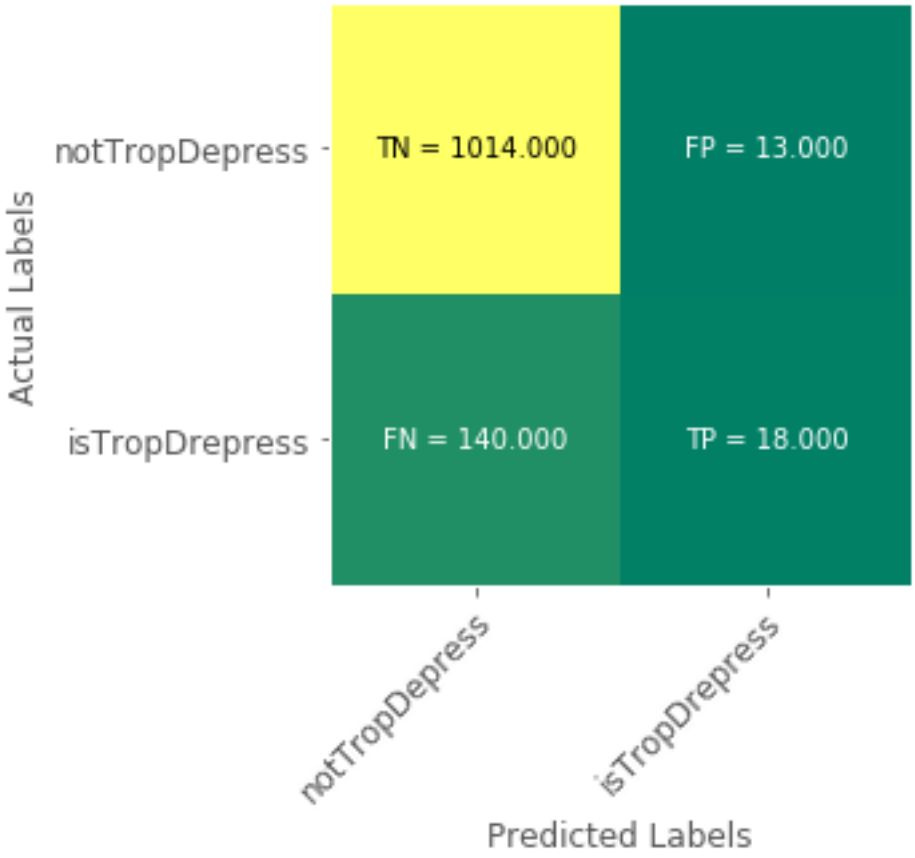
6 DECISION TREE CLASSIFIER

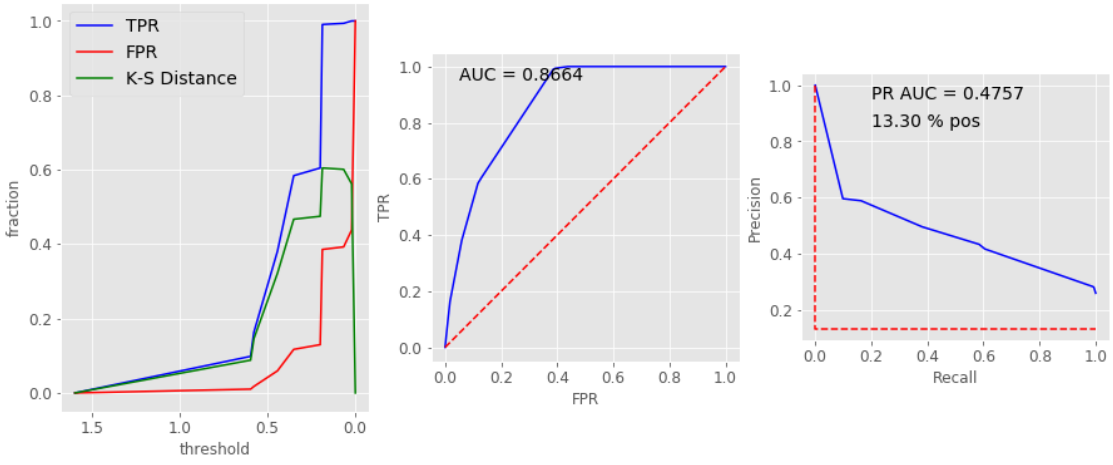
[230]:
```
""" PROVIDED
Create and train DecisionTree for comparision with the ensemble methods
"""
tree_clf = DecisionTreeClassifier(max_depth=200, max_leaf_nodes=10)
tree_clf.fit(Xtrain, ytrain)
```

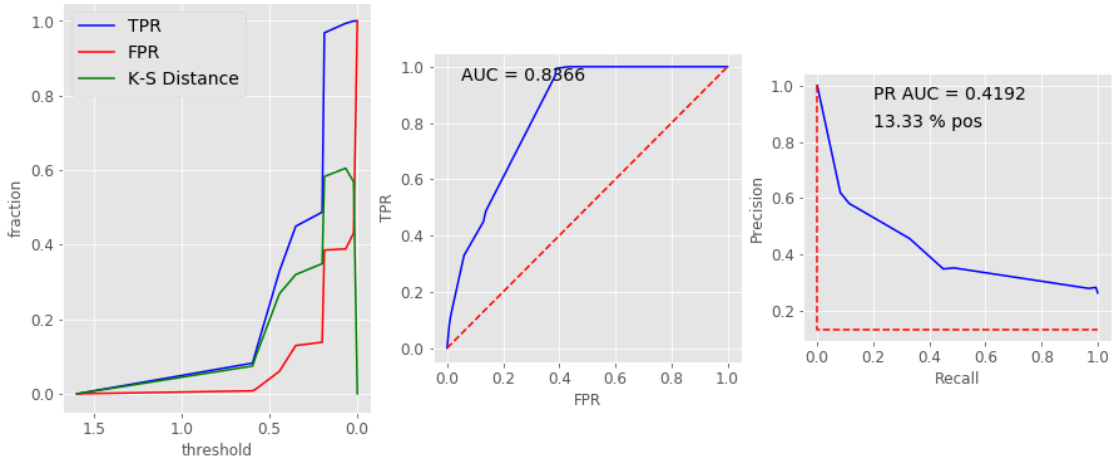
[230]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=200,
            max_features=None, max_leaf_nodes=10,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

[231]:
```
""" 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])
```

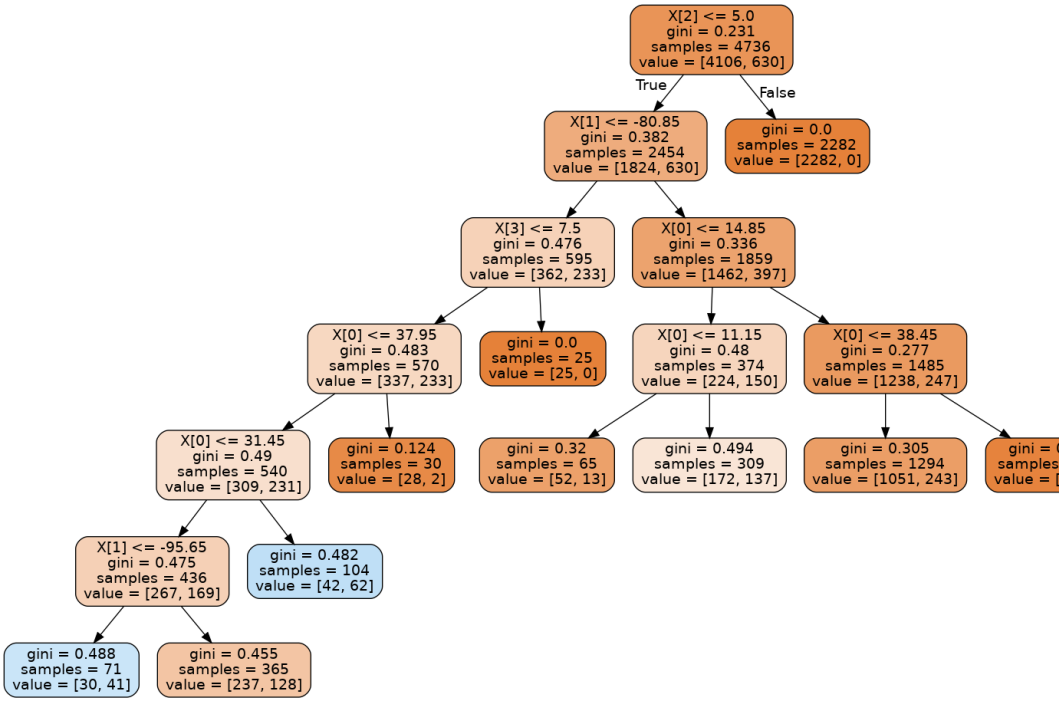

notTropDepress
isTropDrepress
Actual Labels
TN = 4034.000
FP = 72.000
FN = 527.000
TP = 103.000
notTropDepress
isTropDrepress
Predicted Labels


TN = 1014.000
FP = 13.000
FN = 140.000
TP = 18.000
notTropDepress
isTropDrepress
Actual Labels
notTropDepress
isTropDrepress
Predicted Labels


TPR
FPR
K-S Distance
fraction
threshold
AUC = 0.8664
TPR
FPR
PR AUC = 0.4757
13.30 % pos
Precision
Recall



[232]:
```
""" 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)
!dot -Tpng {fname} > {pngname}
```



#![Best Model](tree.png)

7 RANDOM FOREST CLASSIFIER

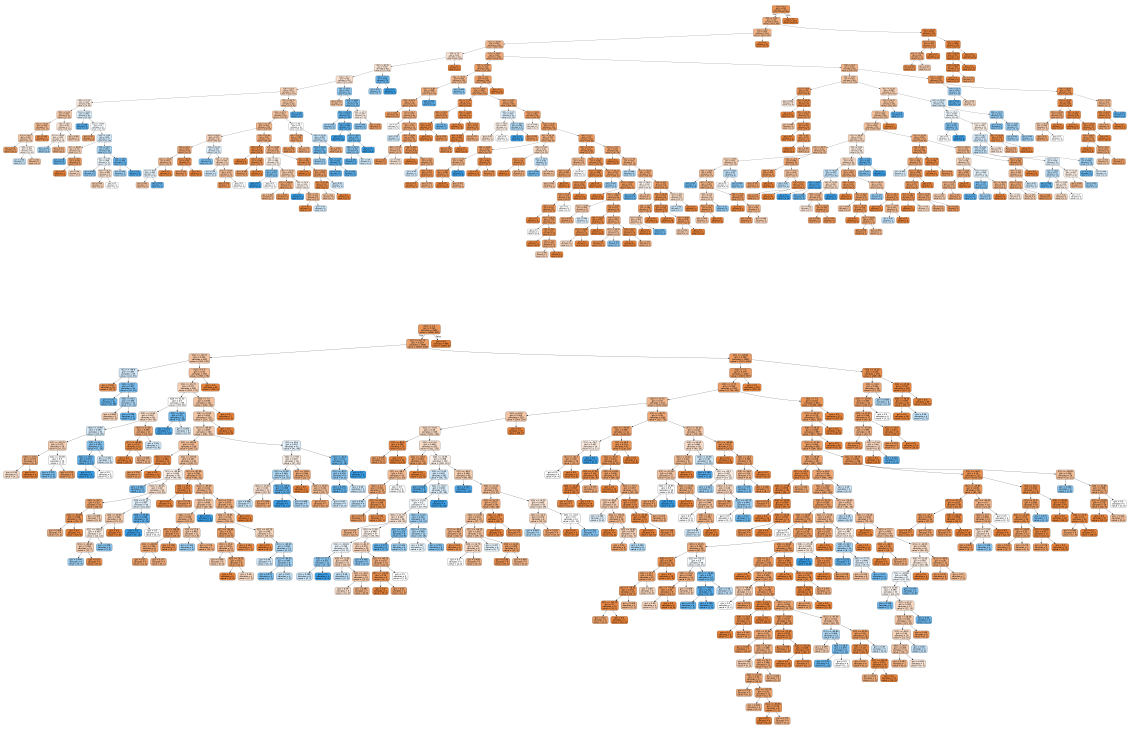
[233]:
```
""" TODO
Create and train RandomForests
Explore various configurations of the hyper-parameters.
Train the models on the training set and evaluate them for the training and
validation sets.
Take a look at the API and the book for the meaning and impact of different
hyper-parameters
"""

forest_clf = RandomForestClassifier(n_estimators=40, max_depth = 70,␣
 ↪min_samples_leaf=4)
forest_clf.fit(Xtrain, ytrain)
```

[233]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=70, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=4, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=40, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

[234]:
```
""" 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
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
```

8 trees

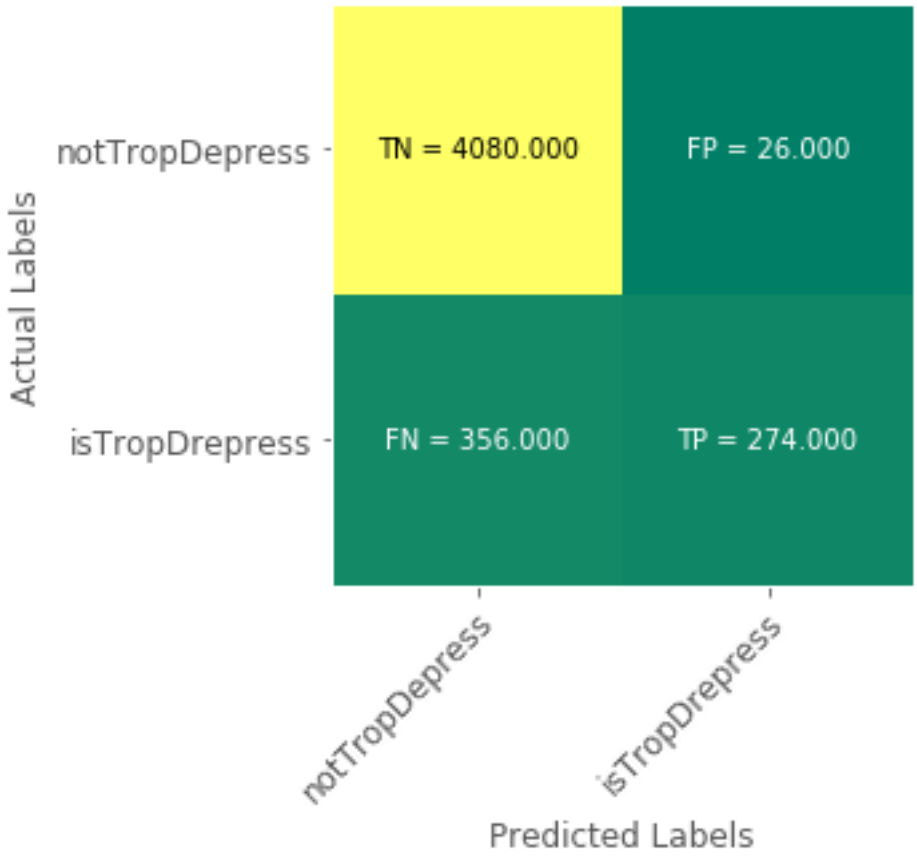


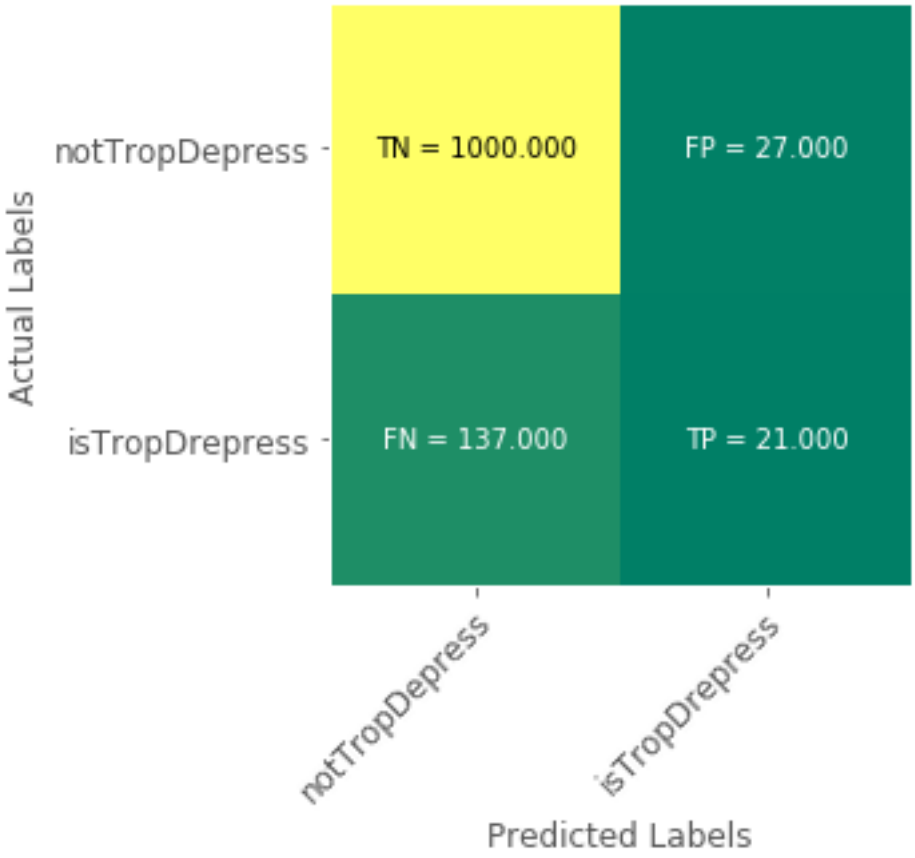
8.0.1 TRAINING AND VALIDATION RESULTS

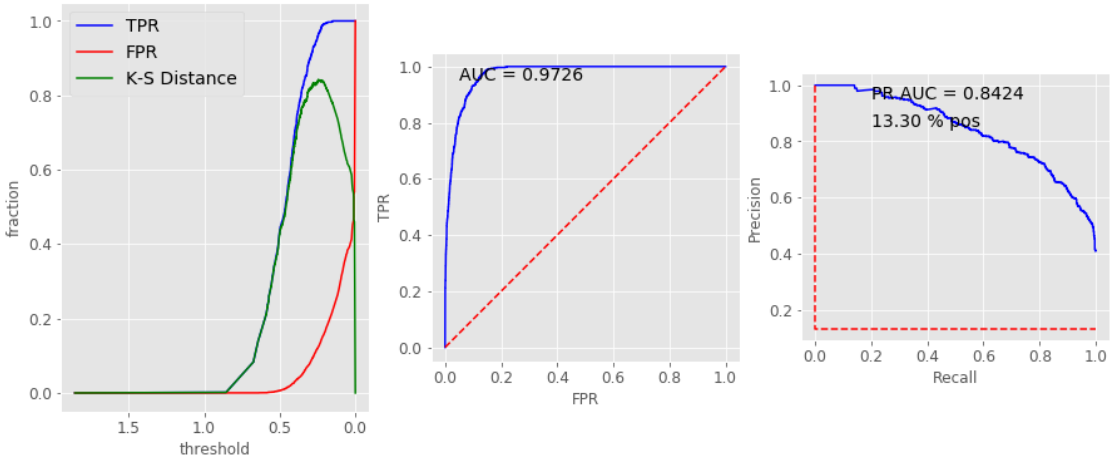
[235]:
```
""" TODO
Compute the predictions, prediction probabilities, and the accuracy scores
for the training and validation sets for the learned instance of the model
"""
# TODO: Compute the model's predictions. use model.predict()
dt_preds2 = forest_clf.predict(Xtrain)
dt_preds_val2 = forest_clf.predict(Xval)


# TODO: Compute the prediction probabilities. use model.predict_proba()
dt_proba2 = forest_clf.predict_proba(Xtrain)
dt_proba_val2 = forest_clf.predict_proba(Xval)


# TODO: Compute the model's mean accuracy. use model.score()
dt_score2 = forest_clf.score(Xtrain, ytrain)
dt_score_val2 = forest_clf.score(Xval, yval)
```

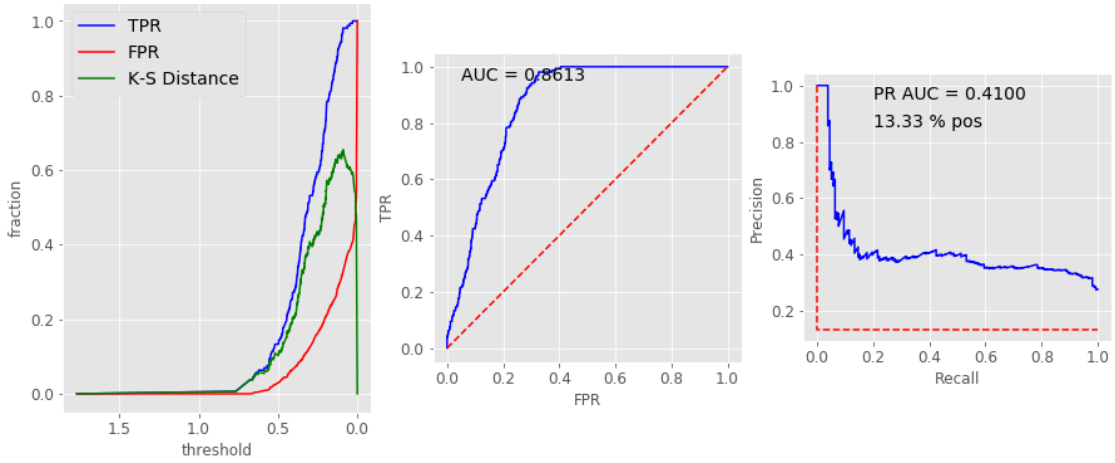
[236]:
```
""" TODO
Display the confusion matrix, KS plot, ROC curve, and PR curve for the training
and validation sets using metrics_plots.ks_roc_prc_plot
```

```
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
dt_cmtx2 = confusion_matrix(ytrain, dt_preds2)
dt_cmtx_val2 = confusion_matrix(yval, dt_preds_val2)
metrics_plots.confusion_mtx_colormap(dt_cmtx2, targetnames, targetnames)
metrics_plots.confusion_mtx_colormap(dt_cmtx_val2, targetnames, targetnames)


# TODO: KS, ROC, and PRC Curves
dt_roc_prc_results2 = metrics_plots.ks_roc_prc_plot(ytrain, dt_proba2[:,1])
dt_roc_prc_results_val2 = metrics_plots.ks_roc_prc_plot(yval, dt_proba_val2[:
↪,1])
```




notTropDepress
isTropDrepress
Actual Labels
TN = 1000.000
FP = 27.000
FN = 137.000
TP = 21.000
notTropDepress
isTropDrepress
Predicted Labels


TPR
FPR
K-S Distance
fraction
threshold
AUC = 0.9726
TPR
FPR
PR AUC = 0.8424
13.30 % pos
Precision
Recall



9 ADABOOSTING

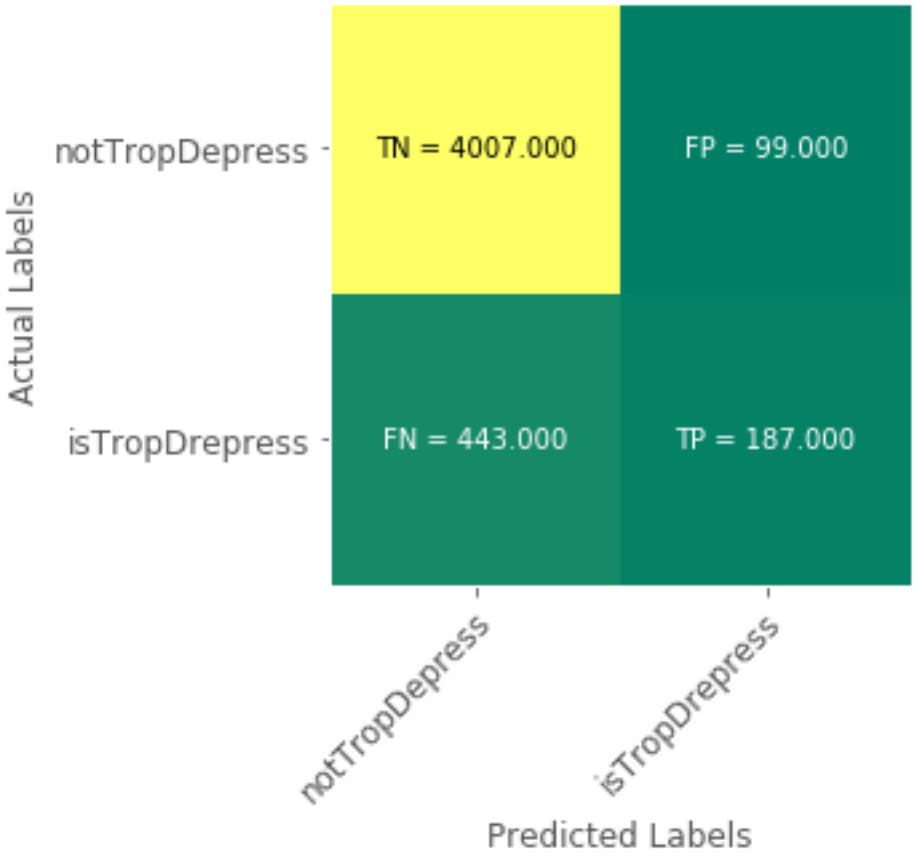
[237]:
```
""" TODO
Create and train a Boosting model
Explore various boosting models to improve your validation performance
Train the models on the training set and evaluate them for the training and
validation sets. Try boosting the benmark tree_clf
"""

ada_clf = AdaBoostClassifier(tree_clf,
                         n_estimators=20,
                         learning_rate=0.08)
ada_clf.fit(Xtrain, ytrain)
```

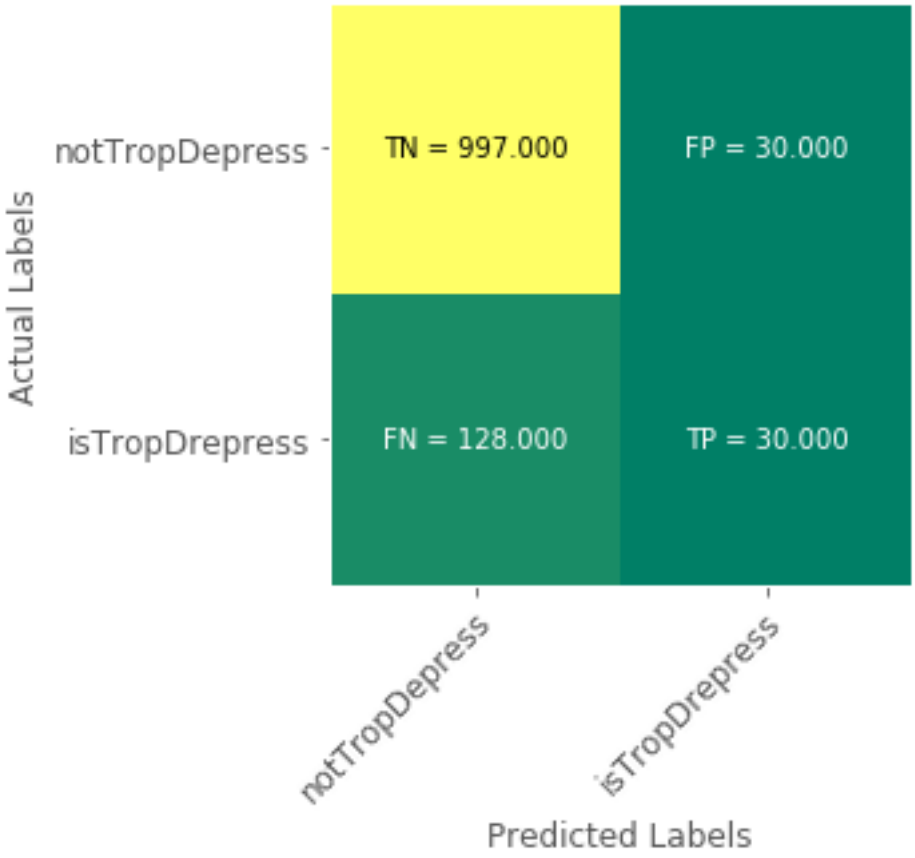
[237]:
```
AdaBoostClassifier(algorithm='SAMME.R',
          base_estimator=DecisionTreeClassifier(class_weight=None,
criterion='gini', max_depth=200,
            max_features=None, max_leaf_nodes=10,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best'),
          learning_rate=0.08, n_estimators=20, random_state=None)
```

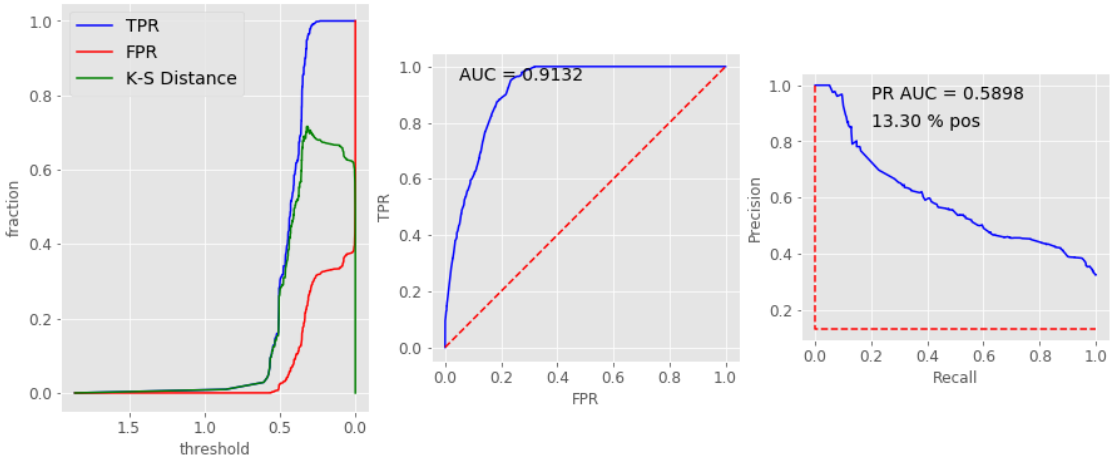
9.0.1 TRAINING AND VALIDATION RESULTS

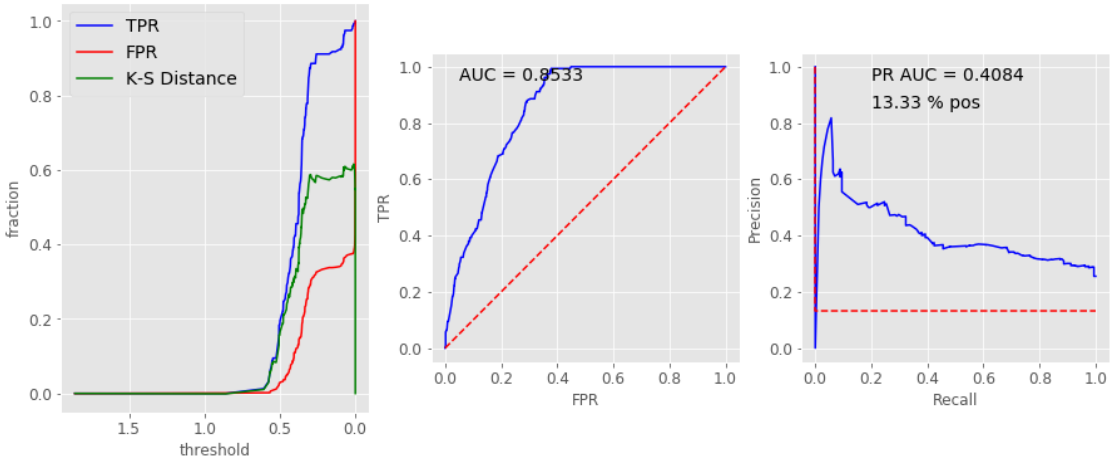
[238]:
```
""" TODO
Compute the predictions, prediction probabilities, and the accuracy scores
for the trianing and validation sets
"""
# TODO: Compute the model's predictions
dt_preds3 = ada_clf.predict(Xtrain)
dt_preds_val3 = ada_clf.predict(Xval)


# TODO: Compute the prediction probabilities
dt_proba3 = ada_clf.predict_proba(Xtrain)
dt_proba_val3 = ada_clf.predict_proba(Xval)


# TODO: Compute the model's scores
dt_score3 = ada_clf.score(Xtrain, ytrain)
dt_score_val3 = ada_clf.score(Xval, yval)
```

[239]:
```
""" 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
dt_cmtx3 = confusion_matrix(ytrain, dt_preds3)
dt_cmtx_val3 = confusion_matrix(yval, dt_preds_val3)
metrics_plots.confusion_mtx_colormap(dt_cmtx3, targetnames, targetnames)
metrics_plots.confusion_mtx_colormap(dt_cmtx_val3, targetnames, targetnames)


# TODO: KS, ROC, and PRC Curves
dt_roc_prc_results3 = metrics_plots.ks_roc_prc_plot(ytrain, dt_proba3[:,1])
dt_roc_prc_results_val3 = metrics_plots.ks_roc_prc_plot(yval, dt_proba_val3[:
 ↪,1])
```


notTropDepress
isTropDrepress
Actual Labels
TN = 4007.000
FP = 99.000
FN = 443.000
TP = 187.000
notTropDepress
isTropDrepress
Predicted Labels


TN = 997.000
FP = 30.000
FN = 128.000
TP = 30.000
notTropDepress
isTropDrepress
Actual Labels
notTropDepress
isTropDrepress
Predicted Labels


TPR
FPR
K-S Distance
fraction
threshold
AUC = 0.9132
TPR
FPR
PR AUC = 0.5898
13.30 % pos
Precision
Recall



10 FEATURE IMPORTANCE

[240]:
```
""" TODO
Display the feature imporantances
see the API for RandomForests and boosted tree
you can create a DataFrame to help with the display
"""
df = pd.DataFrame({'ada_boost': ada_clf.feature_importances_, 'random_forest':␣
↪forest_clf.feature_importances_})
df.index = ['Latitude','Longitude','Low Wind SW','Moderate Wind NE','Moderate␣
↪Wind SE','High Wind NW']
df
```

[240]:

	ada_boost	random_forest
Latitude	0.502707	0.349895
Longitude	0.416717	0.317268
Low Wind SW	0.061839	0.180860
Moderate Wind NE	0.018737	0.093979
Moderate Wind SE	0.000000	0.051703
High Wind NW	0.000000	0.006296

11 DISCUSSION

1. In 2 to 4 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 to 2 paragraphs

My results for randomforest are an improvement over my results for the decisiontree model. This makes sense since reading into randomforest shows that this model is made up of many decision trees in order to compound the best model. I found that tuning some of the hyper parameters increased my accuracy scores as well. For instance, I started off with my n_estimators quite low. This gave me lower PR AUC's for both my training data and testing data. Since I found FPR to always be quite strong with the RandomForest, I tried to get my PR AUC – particularly for the testing data – as high as possible. I found improvement here by tuning the hyperparameters in such a way where the min_sample_leaf size was at 4, the max_depth was at 70, and the n_estimators was at 40. Any significant change to these hurt my PR AUC performance on testing data. There existed a way to maximize the score on my PR AUC for training data to be much higher than I got it, but this hurt the performance of my testing data. I believe this occurred because my model began to overfit. Even now it is overfit because it is not great at positively predicting a positive tropical storm.

In determining the best hyperparameters to use, I basically just guessed and tried things out. I don't have a lot of experience in messing with the hyperparameters for randomforest classification so I tried out things that worked and looked at the sklearn documentation. Moving things up and down did affect performance, and like I mentioned above, my goal was to maximize the recall for PR AUC since this score most close corresponds to how well my model will actual do at predicting whether the weather is going to lead to a tropical storm.

The hyperparameters I looked at were n_estimators, which is the number of trees in the forest. This feature of randomforest allows it to be more accurate potentially than a decision tree. I also looked at max_depth which is the maximum depth of the tree. I found here than a number too small or too large negatively impacted my accuracy. Finally, I focused on min_samples_leaf. This parameters is the minimum number of samples required to be at a leaf node. This parameter seemed to improve my results the smaller it was. The documentation says that the parameter might be even more effective when used for regression.

AdaBoostClassifier adds interesting support to my random forest classifier. Ada boost takes my random forest model and then, according the the sklearn documentation for AdaBoostClassifier, it "fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases. I found that through hyperparameter ada boost tuning I was able to further improve the accuracy of my classification model.

Through the ada boost I worked with hyperparameters to tune the model. The ones I focused on were n_estimators and learning_rate. Adjusting these, I increased my PR AUC for testing only slightly, but had to sacrifice validation performance. The bias-variance tradeoff really shows in this notebook and the art in adjusting this so much as to not overfit or underfit is something I am learning by analyzing the metrics. It is also interesting to note the difference in feature importance between ada boost and random forest. Ada boost figures wind speed to be either negligible or not important, but random forest is more inclined to consider the features. Such interesting metrics to analyze!

[ ]: