# feateng_model

September 17, 2019

[17]:
```
# Import libraries

import re

%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

import numpy as np

import pandas as pd

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import multilabel_confusion_matrix, accuracy_score,␣
 ↪classification_report

from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestClassifier

import xgboost as xgb
import lightgbm as lgb
```

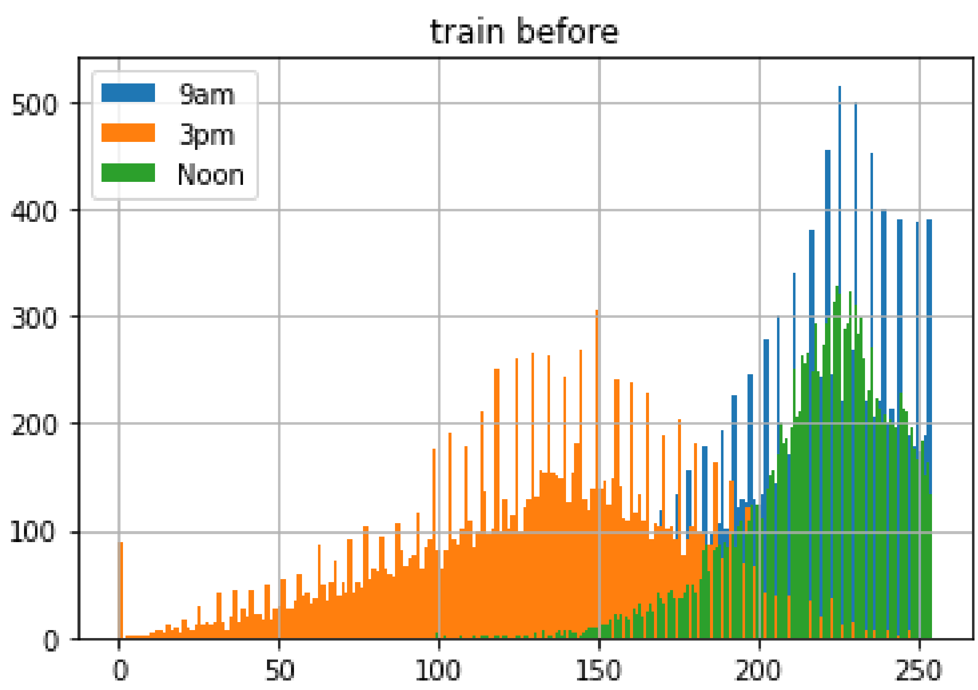
[2]:
```
# Load datasets

TRAIN_FILEPATH = 'data/train.csv'
train_df = pd.read_csv( TRAIN_FILEPATH, header=0 )
display(train_df.shape)

TEST_FILEPATH = 'data/test.csv'
test_df = pd.read_csv( TEST_FILEPATH, header=0 )
display(test_df.shape)
```

```
(15120, 56)
```

```
(565892, 55)
```

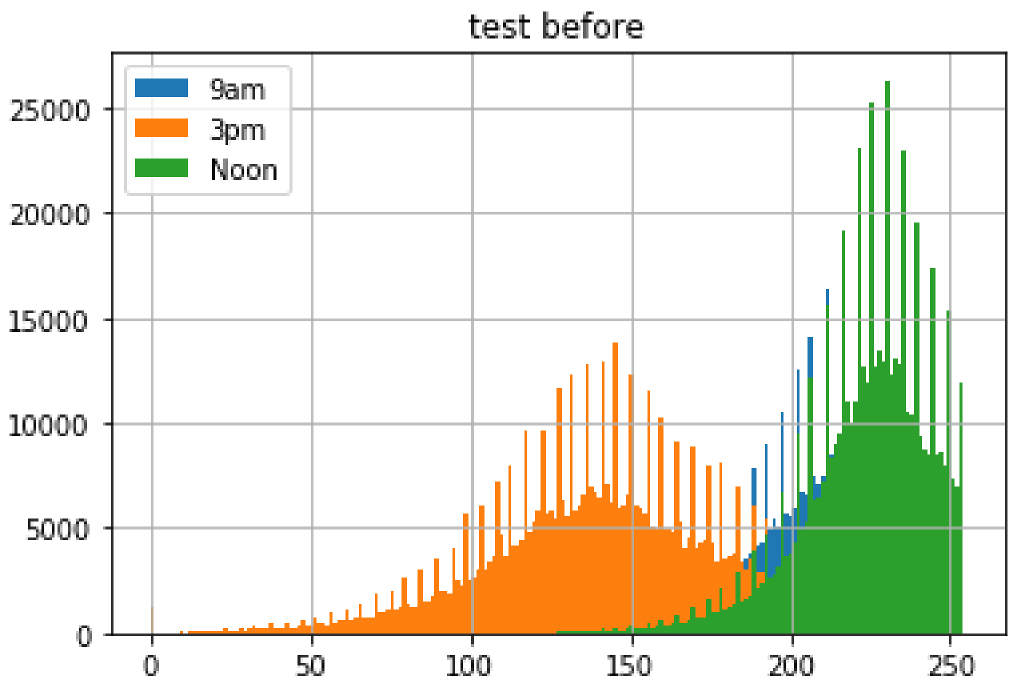
[3]:
```python
# Fix values

# 1. As we can see, some of 'Hillshade_3pm' values = 0 -> fix this
train_df['Hillshade_9am'].hist(bins=200, label='9am')
train_df['Hillshade_3pm'].hist(bins=200, label='3pm')
train_df['Hillshade_Noon'].hist(bins=200, label='Noon')
plt.title('train before')
plt.legend()
plt.show()

display(
    'train before:',
    train_df[ train_df['Hillshade_3pm'] < 5 ]['Hillshade_3pm'].value_counts()
)

test_df['Hillshade_9am'].hist(bins=200, label='9am')
test_df['Hillshade_3pm'].hist(bins=200, label='3pm')
test_df['Hillshade_Noon'].hist(bins=200, label='Noon')
plt.title('test before')
plt.legend()
plt.show()

display(
    'test before',
    test_df[ test_df['Hillshade_3pm'] < 5 ]['Hillshade_3pm'].value_counts()
)

# Use data from test set (500k rows), forget about train set (15k)
hillshade_pred_useful_columns = [
    col_name for col_name
    in test_df.columns.values
    if col_name not in ['Hillshade_3pm', 'Id', 'Cover_Type']
]
train_topredict_rows = train_df[ train_df['Hillshade_3pm'] == 0␣
 ↪][hillshade_pred_useful_columns]
test_topredict_rows = test_df[ test_df['Hillshade_3pm'] == 0␣
 ↪][hillshade_pred_useful_columns]

totrain_rows_X = test_df[ test_df['Hillshade_3pm'] != 0␣
 ↪][hillshade_pred_useful_columns]
totrain_rows_y = test_df[ test_df['Hillshade_3pm'] != 0 ]['Hillshade_3pm']

predictor = KNeighborsRegressor(n_neighbors=10)  # todo: estimate accuracy
predictor.fit( totrain_rows_X, totrain_rows_y )

train_df.loc[ train_topredict_rows.index.values, 'Hillshade_3pm' ] = np.around(␣
 ↪predictor.predict(train_topredict_rows) )
```

```
test_df.loc[ test_topredict_rows.index.values, 'Hillshade_3pm' ] = np.around(␣
↪predictor.predict(test_topredict_rows) )

# 'after' result :
plt.title('train after')
train_df['Hillshade_9am'].hist(bins=200, label='9am')
train_df['Hillshade_3pm'].hist(bins=200, label='3pm')
train_df['Hillshade_Noon'].hist(bins=200, label='Noon')
plt.legend()
plt.show()

display(
    'train after:',
    train_df[ train_df['Hillshade_3pm'] < 5 ]['Hillshade_3pm'].value_counts()
)

plt.title('test after')
test_df['Hillshade_9am'].hist(bins=200, label='9am')
test_df['Hillshade_3pm'].hist(bins=200, label='3pm')
test_df['Hillshade_Noon'].hist(bins=200, label='Noon')
plt.legend()
plt.show()

display(
    'test after',
    test_df[ test_df['Hillshade_3pm'] < 5 ]['Hillshade_3pm'].value_counts()
)
```



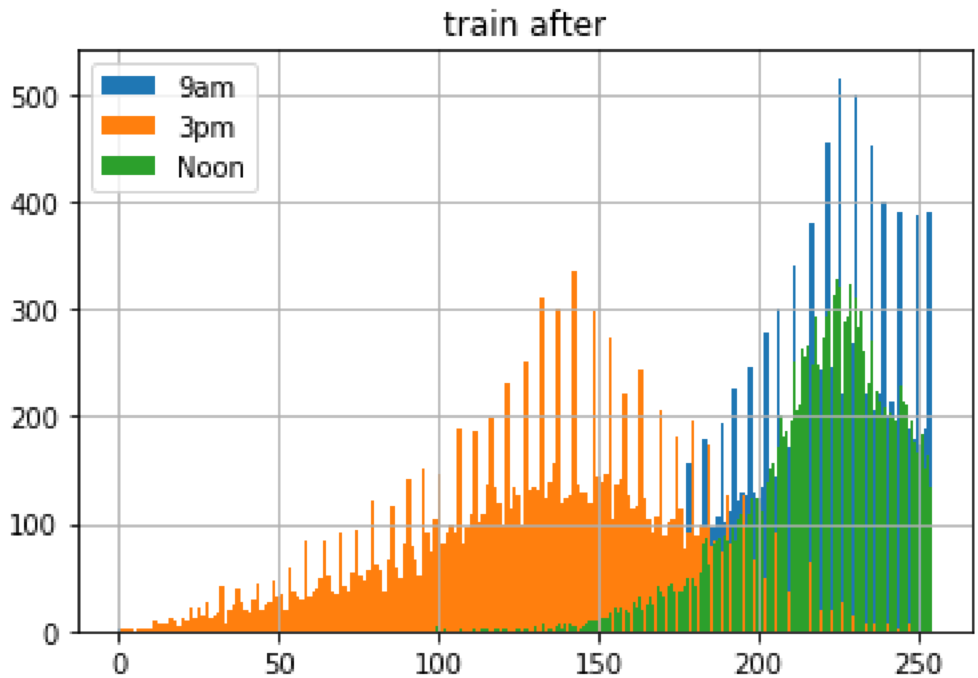
'train before:'

```
0    88
3     3
4     1
1     1
Name: Hillshade_3pm, dtype: int64
```

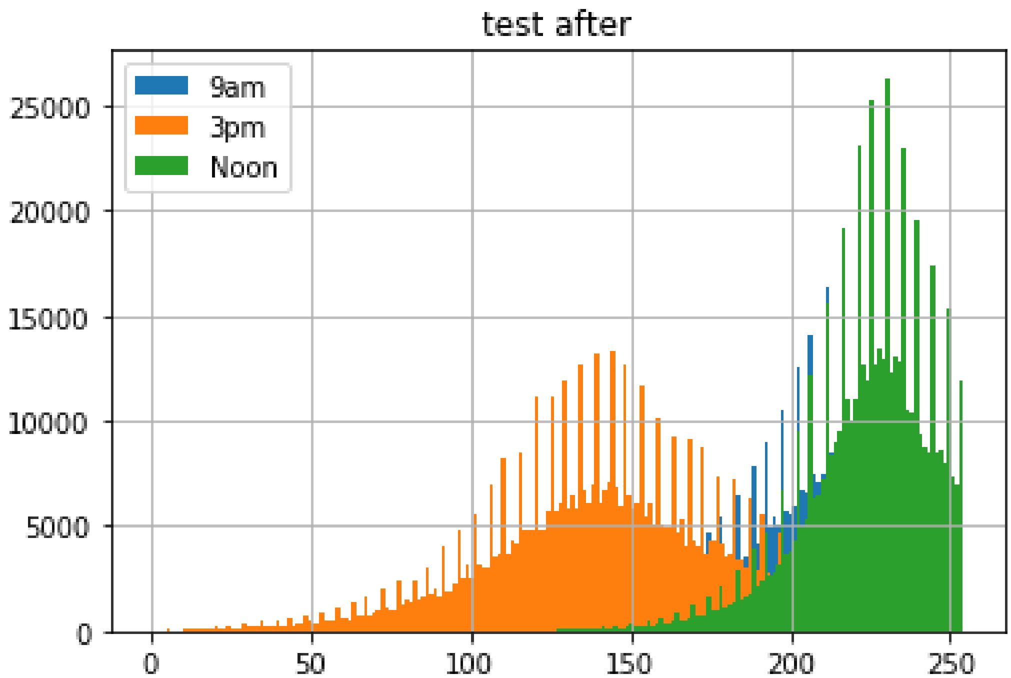



'test before'

```
0    1250
4      19
2      15
1      14
3      12
Name: Hillshade_3pm, dtype: int64
```




```
'train after:'


3.0    3
4.0    1
1.0    1
Name: Hillshade_3pm, dtype: int64
```




```
'test after'
```

```
4.0    19
2.0    15
1.0    14
3.0    12
Name: Hillshade_3pm, dtype: int64
```

[4]:
```
# Work with concatenated feautures

traintest_df = pd.concat( [train_df, test_df], sort=False, ignore_index=True )
```

[5]:
```
# Combine Soil_TypeX and Wilderness_AreaX into single feature

def merge_onehot( dataset_df, col_name_no_x ):
    """Convert col_nameX features to single feature
    X means some integer value.
    Doesn't work with multiple calls - returns 0s for all col_name_no_x.
    """
    dataset_df_cpy = dataset_df.copy()
    # 1. Identify columns
    all_df_columns = dataset_df_cpy.columns.values
    re_pattern_compiled = re.compile( "^{0}(\d+)$".format( col_name_no_x ) )
```

```
    matched_columns = list(filter( re_pattern_compiled.match, all_df_columns ))
    # 2. Change columns: multiply by 'x' value
    for matched_column in matched_columns:
        col_name_x_value = re_pattern_compiled.match(matched_column).groups()[0]
        dataset_df_cpy[matched_column] *= int( col_name_x_value )
    # 3. Merge col_namex columns into single col_name column
    dataset_df_cpy[col_name_no_x] = 0
    for matched_column in matched_columns:
        dataset_df_cpy[col_name_no_x] += dataset_df_cpy[matched_column]
    # 4. Drop col_namex columns
    dataset_df_cpy = dataset_df_cpy.drop( matched_columns, axis=1 )
    return dataset_df_cpy

def _ugly_merge_wildernessarea_soiltype_traintest( train_or_test_df ):
    train_or_test_df = merge_onehot( train_or_test_df,␣
 ↪col_name_no_x='Wilderness_Area' )
    train_or_test_df = merge_onehot( train_or_test_df,␣
 ↪col_name_no_x='Soil_Type' )
    return train_or_test_df


# train_df = _ugly_merge_wildernessarea_soiltype_traintest( train_df )
# test_df = _ugly_merge_wildernessarea_soiltype_traintest( test_df )

# display(train_df.shape, test_df.shape)

traintest_df = _ugly_merge_wildernessarea_soiltype_traintest( traintest_df )
```

[6]:
```
# Feature engineering

def add_soil_family_inplace( df ):
    # src: https://www.kaggle.com/c/forest-cover-type-prediction/data
    # (soiltypeX, soiltypeY, ...): family_name_str
    soiltype_family_mapping = {
        (2, 4): 1, # 'ratake',
        (10, 11, 13, 32, 33): 2, # 'catamount',
        (21, 22, 23, 24, 25, 27, 28): 3, # 'leighcan',
        (38, 39, 40): 4, # 'moran'
#         (... all other IDs ...): 'other_type'
    }
    df['soil_family'] = 5  # 'other_type'
    for i in df.index:
        soil_type_value = df.at[i, 'Soil_Type']
        for key in soiltype_family_mapping.keys():
            if soil_type_value in key:
                df.at[i, 'soil_family'] = soiltype_family_mapping[key]
```

```python
def add_soil_complex_inplace( df ):
    # (soiltypeX, soiltypeY, ...): complex_name_str
    soiltype_complex_mapping = {
        (1, 3, 4, 5, 6, 10, 27, 28, 33): 1, # 'rock_outcrop',
        (11, 12, 34, 40): 2, # 'rock_land',
        (20, 23): 3, # 'typic_cryaquolls',
        (26, 31): 4, # 'catamaount_families',
        (29, 30): 5, # 'legault_family',
        (32, 39): 6, # 'leighcan_family',
    }
    df['soil_complex'] = 7  # 'other_type'
    for i in df.index:
        soil_type_value = df.at[i, 'Soil_Type']
        for key in soiltype_complex_mapping.keys():
            if soil_type_value in key:
                df.at[i, 'soil_complex'] = soiltype_complex_mapping[key]


def add_soil_stonetype_inplace( df ):
    # (soiltypeX, soiltypeY, ...): stonetype_name_str
    soiltype_stonetype_mapping = {
        (1, 2, 6, 9, 12, 18, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36,␣
↪37, 38, 39, 40): 1, # 'stony',
        (3, 4, 5, 10, 11, 13, ): 2, # 'rubbly'
    }
    df['soil_stonetype'] = 3 # 'other_type'
    for i in df.index:
        soil_type_value = df.at[i, 'Soil_Type']
        for key in soiltype_stonetype_mapping.keys():
            if soil_type_value in key:
                df.at[i, 'soil_stonetype'] = soiltype_stonetype_mapping[key]


def main_feature_engineering( df ):
    df_cpy = df.copy()

    # Median hillshade index [0-255]
    df_cpy['median_hillshade_idx'] = df_cpy[['Hillshade_9am', 'Hillshade_Noon',␣
↪'Hillshade_3pm']].agg('median', axis='columns')
    # Mean hillshade index [0-255]
    df_cpy['median_hillshade_idx'] = df_cpy[['Hillshade_9am', 'Hillshade_Noon',␣
↪'Hillshade_3pm']].agg('mean', axis='columns')
    # Distance to hydrology (using HorizontalDistance and VerticalDistance to␣
↪hydrology) - pythagoras theorem
```

```
    df_cpy['hydrology_distance'] = np.sqrt(␣
 ↪df_cpy['Horizontal_Distance_To_Hydrology']**2 +␣
 ↪df_cpy['Vertical_Distance_To_Hydrology']**2 )
    # Aspect binning: 20 intervals
    ASPECT_BINS_CNT = 20
    df_cpy['aspect_bin'] = pd.cut( df_cpy['Aspect'], bins=ASPECT_BINS_CNT )
    df_cpy['aspect_bin'] = df_cpy['aspect_bin'].apply(
        lambda interval: '{0}_{1}'.format(interval.left, interval.right)
    )
    # Polynomial features: dependance of 9am->noon->3pm->9am
    df_cpy['9am_noon_dep'] = df_cpy['Hillshade_9am'] * df_cpy['Hillshade_Noon']
    df_cpy['noon_3pm_dep'] = df_cpy['Hillshade_Noon'] * df_cpy['Hillshade_3pm']
    df_cpy['3pm_9am_dep'] = df_cpy['Hillshade_3pm'] * df_cpy['Hillshade_9am']
    # Cosine of slope: relationships between hillshade and other features
    df_cpy['slope_cosine'] = np.cos( df_cpy['Slope'] )
    # Log-transform 'Elevation' feature
    df_cpy['Elevation'] = np.log1p( df_cpy['Elevation'] )


    return df_cpy


def apply_feature_engineering( df ):
    print('soil family...')
    add_soil_family_inplace( df )
    print('soil complex...')
    add_soil_complex_inplace( df )
    print('soil stonetype...')
    add_soil_stonetype_inplace( df )
    print('main feature engineering...')
    df = main_feature_engineering( df )
    return df


# train_df = apply_feature_engineering( train_df )
# test_df = apply_feature_engineering( test_df )

traintest_df = apply_feature_engineering( traintest_df )
```

```
soil family...
soil complex...
soil stonetype...
main feature engineering...
```

[7]:
```
# 29th - x3
# 15 - only 3 values
```

```
# display(
#     traintest_df['Soil_Type'].value_counts()
# )
# traintest_df.shape
```

[8]:

```
# Convert 'aspect_bin' to numerical format

aspectbin_lblencoder = LabelEncoder()
traintest_df['aspect_bin'] = aspectbin_lblencoder.fit_transform(␣
↪traintest_df['aspect_bin'] )
```

[9]:

```
# Split traintest df for model validation

train = traintest_df.iloc[:train_df.shape[0], :]
test = traintest_df.iloc[train_df.shape[0]:, :]

# remove redundant columns
train_labels = train['Cover_Type']
train = train.drop( ['Id', 'Cover_Type'], axis=1 )
test = test.drop( ['Id'], axis=1 )

# train-validation split
VALIDATION_SIZE = 0.3
X_tr, X_val, y_tr, y_val = train_test_split(
    train, train_labels,
    test_size=VALIDATION_SIZE,
    shuffle=True
)

display(X_tr.shape, X_val.shape)
```

```
(10584, 22)
```

```
(4536, 22)
```

[10]:

```
# 1. Try out lgbm

lgb_model = lgb.LGBMClassifier(
    learning_rate=0.25,
    max_depth=-1,
    n_estimators=1000,
    objective='multiclass',
    n_jobs=8,
    verbose=1
)
lgb_model.fit( X_tr, y_tr )
lgb_y_val_pred = lgb_model.predict( X_val )
```

```python
display( accuracy_score(y_val, lgb_y_val_pred) )
print( classification_report(y_val, lgb_y_val_pred) )
display( multilabel_confusion_matrix(y_val, lgb_y_val_pred) )
```

```
0.8608906525573192
```

```
              precision    recall  f1-score   support

         1.0       0.77      0.71      0.74       632
         2.0       0.74      0.70      0.72       638
         3.0       0.85      0.83      0.84       613
         4.0       0.96      0.96      0.96       667
         5.0       0.91      0.94      0.93       686
         6.0       0.84      0.90      0.86       645
         7.0       0.93      0.98      0.95       655

    accuracy                           0.86      4536
   macro avg       0.86      0.86      0.86      4536
weighted avg       0.86      0.86      0.86      4536
```

```
array([[[3772,  132],
        [ 182,  450]],

       [[3741,  157],
        [ 192,  446]],

       [[3835,   88],
        [ 104,  509]],

       [[3839,   30],
        [  29,  638]],

       [[3789,   61],
        [  42,  644]],

       [[3777,  114],
        [  67,  578]],

       [[3832,   49],
        [  15,  640]]])
```

[11]:
```python
display(
    pd.DataFrame({
        'feature_name': X_tr.columns.values,
```

```
        'feature_imp': lgb_model.feature_importances_
    }).sort_values(by='feature_imp', ascending=False)
)
```

```
                         feature_name  feature_imp
9   Horizontal_Distance_To_Fire_Points        16296
5      Horizontal_Distance_To_Roadways        16148
0                           Elevation        14868
4      Vertical_Distance_To_Hydrology         9797
16                  hydrology_distance         7317
18                        9am_noon_dep         7295
1                              Aspect         6448
3     Horizontal_Distance_To_Hydrology         5268
6                        Hillshade_9am         5238
20                         3pm_9am_dep         4951
21                        slope_cosine         4615
19                        noon_3pm_dep         4501
11                           Soil_Type         4478
15                median_hillshade_idx         4457
8                        Hillshade_3pm         4255
7                       Hillshade_Noon         4034
2                               Slope         3358
17                          aspect_bin         1974
13                        soil_complex         1968
10                     Wilderness_Area         1759
12                         soil_family         1565
14                      soil_stonetype          891
```

[12]:
```
# 2. Try out xgb

xgb_model = xgb.XGBClassifier(
    gamma=0.03,
    learning_rate=0.2,
    max_depth=5,
    n_estimators=1000,
    objective='multi:softmax',
    n_jobs=4
)
xgb_model.fit( X_tr, y_tr )
xgb_y_val_pred = xgb_model.predict( X_val )

display( accuracy_score(y_val, xgb_y_val_pred) )
print( classification_report(y_val, xgb_y_val_pred) )
display( multilabel_confusion_matrix(y_val, xgb_y_val_pred) )
```

```
0.8474426807760141
```

```
              precision    recall  f1-score   support

         1.0       0.76      0.70      0.73       632
         2.0       0.72      0.68      0.70       638
         3.0       0.83      0.81      0.82       613
         4.0       0.95      0.96      0.95       667
         5.0       0.91      0.93      0.92       686
         6.0       0.81      0.87      0.84       645
         7.0       0.94      0.97      0.95       655

    accuracy                           0.85      4536
   macro avg       0.84      0.84      0.84      4536
weighted avg       0.84      0.85      0.85      4536
```

```
array([[[3766,  138],
        [ 191,  441]],

       [[3724,  174],
        [ 201,  437]],

       [[3819,  104],
        [ 119,  494]],

       [[3832,   37],
        [  29,  638]],

       [[3784,   66],
        [  49,  637]],

       [[3762,  129],
        [  86,  559]],

       [[3837,   44],
        [  17,  638]]])
```

[13]:
```
display(
    pd.DataFrame({
        'feature_name': X_tr.columns.values,
        'feature_imp': xgb_model.feature_importances_
    }).sort_values(by='feature_imp', ascending=False)
)
```

| | feature_name | feature_imp |
|---|---|---|
| 11 | Soil_Type | 0.206622 |
| 0 | Elevation | 0.180959 |
| 10 | Wilderness_Area | 0.108293 |

```
14                    soil_stonetype    0.071709
12                       soil_family    0.063648
18                      9am_noon_dep    0.043666
3     Horizontal_Distance_To_Hydrology  0.037654
5      Horizontal_Distance_To_Roadways  0.027700
9   Horizontal_Distance_To_Fire_Points  0.025955
13                      soil_complex    0.025126
6                        Hillshade_9am  0.024648
16                  hydrology_distance  0.022914
7                       Hillshade_Noon  0.022648
1                               Aspect  0.018521
8                        Hillshade_3pm  0.018179
4       Vertical_Distance_To_Hydrology  0.018095
2                                Slope  0.017150
19                      noon_3pm_dep    0.014901
20                       3pm_9am_dep    0.014727
17                        aspect_bin    0.013576
15               median_hillshade_idx   0.013010
21                      slope_cosine    0.010297
```

[14]:
```python
# 3. Try out rfc

rfc_model = RandomForestClassifier(
    n_estimators=1000,
    n_jobs=-1
)
rfc_model.fit( X_tr, y_tr )
rfc_y_val_pred = rfc_model.predict( X_val )

display( accuracy_score(y_val, rfc_y_val_pred) )
print( classification_report(y_val, rfc_y_val_pred) )
display( multilabel_confusion_matrix(y_val, rfc_y_val_pred) )
```

```
0.8441358024691358
```

```
              precision    recall  f1-score   support

         1.0       0.75      0.72      0.73       632
         2.0       0.72      0.68      0.70       638
         3.0       0.83      0.79      0.81       613
         4.0       0.93      0.96      0.94       667
         5.0       0.91      0.92      0.91       686
         6.0       0.81      0.87      0.84       645
         7.0       0.92      0.96      0.94       655

    accuracy                           0.84      4536
```

```
   macro avg       0.84      0.84      0.84      4536
weighted avg       0.84      0.84      0.84      4536


array([[[3756,  148],
        [ 180,  452]],

       [[3733,  165],
        [ 206,  432]],

       [[3824,   99],
        [ 129,  484]],

       [[3818,   51],
        [  26,  641]],

       [[3787,   63],
        [  56,  630]],

       [[3763,  128],
        [  86,  559]],

       [[3828,   53],
        [  24,  631]]])
```

[15]:

```
display(
    pd.DataFrame({
        'feature_name': X_tr.columns.values,
        'feature_imp': rfc_model.feature_importances_
    }).sort_values(by='feature_imp', ascending=False)
)
```

| | feature_name | feature_imp |
|---|---|---|
| 0 | Elevation | 0.207208 |
| 11 | Soil_Type | 0.112166 |
| 5 | Horizontal_Distance_To_Roadways | 0.073421 |
| 9 | Horizontal_Distance_To_Fire_Points | 0.057405 |
| 10 | Wilderness_Area | 0.054237 |
| 16 | hydrology_distance | 0.044296 |
| 12 | soil_family | 0.042523 |
| 18 | 9am_noon_dep | 0.040009 |
| 3 | Horizontal_Distance_To_Hydrology | 0.037526 |
| 4 | Vertical_Distance_To_Hydrology | 0.036029 |
| 6 | Hillshade_9am | 0.033354 |
| 13 | soil_complex | 0.033267 |
| 1 | Aspect | 0.030202 |
| 8 | Hillshade_3pm | 0.027031 |

```
19                 noon_3pm_dep     0.026675
20                  3pm_9am_dep     0.025613
7                Hillshade_Noon     0.024566
15         median_hillshade_idx     0.023852
2                         Slope     0.020532
14               soil_stonetype     0.020438
21                 slope_cosine     0.015050
17                   aspect_bin     0.014602
```

[16]:
```
# Try to increase models performance by fixing skewness

# tofix_skew_col_names = []
# for col_name in traintest_df:
#     skew_value = traintest_df[col_name].skew()
#     if not -1 < skew_value < 1:
#         tofix_skew_col_names.append( col_name )

# Cannot apply boxcox1p for all columns
# for col_name in tofix_skew_col_names:
#     try:
#         boxcox_norm = boxcox_normmax( trainteset_df[col_name] + 1 )
#         display( col_name, boxcox1p(traintest_df[col_name], boxcox_norm).
 ↪skew() )
#     except:
#         display('cannot apply boxcox for {0}'.format(col_name))

# cant do that
```

[24]:
```
# Find best hyperparameters for RFC, XGB, LGB classifiers

# 1. RFC
rfc_model = RandomForestClassifier( )
rfc_param_grid = {
    'n_estimators': [100, 250, 500, 1000],
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 4, 5, 10, None],
    'min_samples_split': np.linspace(0.1, 1, 5),
    'max_features': [2, 5, 'auto'],
    'n_jobs': [4],
}
rfc_grid_search = GridSearchCV(
    estimator=rfc_model,
    param_grid=rfc_param_grid,
    cv=5,
    verbose=2, iid=False, n_jobs=4
)
rfc_grid_search.fit( X_tr, y_tr )
```

```
display( rfc_grid_search.best_params_ )
```

```
Fitting 5 folds for each of 600 candidates, totalling 3000 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  33 tasks      | elapsed:   11.9s
[Parallel(n_jobs=4)]: Done 154 tasks      | elapsed:   56.2s
[Parallel(n_jobs=4)]: Done 357 tasks      | elapsed:  2.1min
[Parallel(n_jobs=4)]: Done 640 tasks      | elapsed:  4.0min
[Parallel(n_jobs=4)]: Done 1005 tasks      | elapsed:  6.1min
[Parallel(n_jobs=4)]: Done 1450 tasks      | elapsed:  9.1min
[Parallel(n_jobs=4)]: Done 1977 tasks      | elapsed: 12.6min
[Parallel(n_jobs=4)]: Done 2584 tasks      | elapsed: 16.9min
[Parallel(n_jobs=4)]: Done 3000 out of 3000 | elapsed: 19.9min finished

{'criterion': 'gini',
 'max_depth': None,
 'max_features': 5,
 'min_samples_split': 0.1,
 'n_estimators': 500,
 'n_jobs': 4}
```

[37]:
```
# 2. LGBM

lgb_model = lgb.LGBMClassifier(
    objective='multiclass',
    n_jobs=4, verbose=0
)
lgb_grid_params = {
    'learning_rate': [0.2, 0.25, 0.3],
    'num_leaves': [ int(val) for val in np.linspace(5, 25, 3) ],
    'max_depth': [-1, 5, 15, 25],
    'n_estimators': [100, 250, 500, 1000],
    'min_split_gain': [0.0, 0.05, 0.1]
}
lgb_grid_search = GridSearchCV(
    estimator=lgb_model,
    param_grid=lgb_grid_params,
    cv=2,
    verbose=2, iid=False, n_jobs=4
)
lgb_grid_search.fit( X_tr, y_tr )
display( lgb_grid_search.best_params_ )
```

```
Fitting 2 folds for each of 432 candidates, totalling 864 fits
```

```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  33 tasks      | elapsed: 12.1min
[Parallel(n_jobs=4)]: Done 154 tasks      | elapsed: 66.0min
[Parallel(n_jobs=4)]: Done 357 tasks      | elapsed: 148.4min
[Parallel(n_jobs=4)]: Done 640 tasks      | elapsed: 259.8min
[Parallel(n_jobs=4)]: Done 864 out of 864 | elapsed: 344.6min finished
```

```
{'learning_rate': 0.3,
 'max_depth': -1,
 'min_split_gain': 0.0,
 'n_estimators': 250,
 'num_leaves': 25}
```

[ ]:

```
# 3. XGB

xgb_model = xgb.XGBClassifier(
    objective='multi:softmax',
    n_jobs=4, verbosity=0
)
xgb_grid_params = {
    'gamma': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [3, 5, 25],
    'n_estimators': [250, 500],
}
xgb_grid_search = GridSearchCV(
    estimator=xgb_model,
    param_grid=xgb_grid_params,
    cv=2,
    verbose=2, iid=False, n_jobs=4
)
xgb_grid_search.fit( X_tr, y_tr )
display( xgb_grid_search.best_params_ )
```

```
Fitting 2 folds for each of 24 candidates, totalling 48 fits
```

```
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
```