# VSB Power Line Fault Detection

### Importing required libraries

In [1]:

```
#data structures
import pandas as pd
import numpy as np

from sklearn.model_selection import StratifiedKFold,train_test_split,RandomizedSear
from sklearn.metrics import matthews_corrcoef,make_scorer
from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

from tqdm import tqdm
import ast
import pickle

import warnings
warnings.filterwarnings('ignore')
```

## Training models

In [2]:

```
#loading the spectra features
train_spectra_features = np.load('trainfeatures/train_spectra_features.npy')
```

In [4]:

```
#loading the signal
train_sig_features = np.load('trainfeatures/train_signal_features.npy')
```

In [5]:

```
#loading the train labels
train_labels = np.load('traindata/train_labels.npy')
```

In [6]:

```
#concatinating the train features
X_train_features = np.concatenate((train_spectra_features,train_sig_features),axis=
```

In [7]:

```
X_train_features.shape
```

Out[7]:

```
(2904, 76)
```

## Logistic Regression

In [16]:

```
# split into train and CV data
n_splits = 5
splits = list(StratifiedKFold(n_splits=n_splits, shuffle=True).split(X_train_featur

models = []
scores = np.zeros(n_splits)

print('Training...')
print('MCC training & cv') # MCC = Matthews Correlation Coefficient

alphas = [10**i for i in range(0,5)]
cv_mccs = []
for alpha in alphas:
    print('for alpha = {}'.format(alpha))
    for i, (idx_train, idx_cv) in enumerate(splits):

        #train and cv split
        X_train = X_train_features[idx_train, :]
        y_train = train_labels[idx_train]

        X_cv = X_train_features[idx_cv, :]
        y_cv = train_labels[idx_cv]

        #initalizing and fitting the model
        model =LogisticRegression(C=alpha,max_iter=4000)
        model.fit(X_train, y_train.astype(float))

        #prediction
        y_predict_train = model.predict(X_train)
        y_predict_cv = model.predict(X_cv)

        #calculating mcc metric
        score_train = matthews_corrcoef(y_train, y_predict_train)
        score_cv = matthews_corrcoef(y_cv, y_predict_cv)

        #storing the models
        models.append(model)
        scores[i] = score_cv

        #printing the train and cross validation mcc score
        print('%d %.3f %.3f' % (i, score_train, score_cv))

    #average of all the scores
    print('CV scores %.3f ± %.3f' % (np.mean(scores), np.std(scores)))
    cv_mccs.append(np.mean(scores))
    print()
```

```
Training...
MCC training & cv
for alpha = 1
0 0.307 0.144
1 0.274 0.290
2 0.287 0.280
3 0.265 0.280
4 0.304 0.211
CV scores 0.241 ± 0.056
```

```
for alpha = 10
0 0.562 0.428
1 0.540 0.526
2 0.492 0.451
3 0.543 0.482
4 0.534 0.421
CV scores 0.462 ± 0.039

for alpha = 100
0 0.728 0.540
1 0.685 0.604
2 0.650 0.632
3 0.685 0.666
4 0.671 0.581
CV scores 0.605 ± 0.043

for alpha = 1000
0 0.750 0.610
1 0.725 0.666
2 0.715 0.686
3 0.732 0.680
4 0.731 0.645
CV scores 0.658 ± 0.028

for alpha = 10000
0 0.777 0.645
1 0.752 0.708
2 0.732 0.706
3 0.746 0.688
4 0.756 0.659
CV scores 0.681 ± 0.025
```

In [17]:

```
best_alpha = alphas[np.argmax(cv_mccs)]
```

In [18]:

```
n_splits = 5
splits = list(StratifiedKFold(n_splits=n_splits, shuffle=True).split(X_train_featur

models = []
scores = np.zeros(n_splits)

print('Training...')
print('MCC training & cv') # MCC = Matthews Correlation Coefficient

for i, (idx_train, idx_cv) in enumerate(splits):

    #train and cv split
    X_train = X_train_features[idx_train, :]
    y_train = train_labels[idx_train]

    X_cv = X_train_features[idx_cv, :]
    y_cv = train_labels[idx_cv]

    #initalizing and fitting the model
    model =LogisticRegression(C=best_alpha,max_iter=4000)
    model.fit(X_train, y_train.astype(float))

    #prediction
    y_predict_train = model.predict(X_train)
    y_predict_cv = model.predict(X_cv)

    #calculating mcc metric
    score_train = matthews_corrcoef(y_train, y_predict_train)
    score_cv = matthews_corrcoef(y_cv, y_predict_cv)

    #storing the models
    models.append(model)
    scores[i] = score_cv

    #printing the train and cross validation mcc score
    print('%d %.3f %.3f' % (i, score_train, score_cv))

#average of all the scores
print('CV scores %.3f ± %.3f' % (np.mean(scores), np.std(scores)))
```

```
Training...
MCC training & cv
0 0.758 0.624
1 0.765 0.666
2 0.740 0.719
3 0.744 0.660
4 0.727 0.673
CV scores 0.668 ± 0.031
```

### Random Forest Classifier

In [19]:

```
#creating a scorer of mcc to use for randomsearchcv
mcc_scorer = make_scorer(matthews_corrcoef)
```

In [21]:

```
#using random search cv for finding best hyperparameters
dt_cfl = RandomForestClassifier()
params={'n_estimators' : range(10,500,20),'max_depth': range(1,16,2),'max_features'

rand_cv = RandomizedSearchCV(dt_cfl,param_distributions=params,verbose=0,n_jobs=-1,
rand_cv.fit(X_train_features,train_labels)
```

Out[21]:

```
RandomizedSearchCV(cv=None, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=Non
e,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='aut
o',
                                                    max_leaf_nodes=No
ne,
                                                    max_samples=None,
                                                    min_impurity_decr
ease=0.0,
                                                    min_impurity_spli
t=None,
                                                    min_samples_leaf=
1,
                                                    min_samples_split
=2,
                                                    min_weight_fracti
on_leaf=0.0,
                                                    n_estimators=100,
                                                    n_jobs=None,
                                                    oob_score=False,
                                                    random_state=Non
e,
                                                    verbose=0,
                                                    warm_start=Fals
e),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'max_depth': range(1, 16, 2),
                                        'max_features': range(50, 70,
5),
                                        'n_estimators': range(10, 50
0, 20)},
                   pre_dispatch='2*n_jobs', random_state=None, refit=
True,
                   return_train_score=False,
                   scoring=make_scorer(matthews_corrcoef), verbose=0)
```

In [22]:

```
rand_cv.best_estimator_
```

Out[22]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=No
ne,
                       criterion='gini', max_depth=7, max_features=6
5,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=
None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=27
0,
                       n_jobs=None, oob_score=False, random_state=Non
e,
                       verbose=0, warm_start=False)
```

In [23]:

```
# split into train and CV data
n_splits = 5
splits = list(StratifiedKFold(n_splits=n_splits, shuffle=True).split(X_train_featur

models = []
scores = np.zeros(n_splits)

print('Training...')
print('MCC training & cv') # MCC = Matthews Correlation Coefficient
for i, (idx_train, idx_cv) in enumerate(splits):

    #train and cv split
    X_train = X_train_features[idx_train, :]
    y_train = train_labels[idx_train]

    X_cv = X_train_features[idx_cv, :]
    y_cv = train_labels[idx_cv]

    #initalizing and fitting the model
    model =RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=7, max_features=65,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=270,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
    model.fit(X_train, y_train.astype(float))

    #prediction
    y_predict_train = model.predict(X_train)
    y_predict_cv = model.predict(X_cv)

    #calculating mcc metric
    score_train = matthews_corrcoef(y_train, y_predict_train)
    score_cv = matthews_corrcoef(y_cv, y_predict_cv)

    #storing the models
    models.append(model)
    scores[i] = score_cv

    #printing the train and cross validation mcc score
    print('%d %.3f %.3f' % (i, score_train, score_cv))

#average of all the scores
print('CV scores %.3f ± %.3f' % (np.mean(scores), np.std(scores)))
```

```
Training...
MCC training & cv
0 0.938 0.720
1 0.958 0.739
2 0.950 0.706
3 0.969 0.680
4 0.938 0.659
CV scores 0.701 ± 0.028
```

## LGBM Classifier

In [28]:

```
params = {
 'num_leaves': [31, 127],
 'reg_alpha': [0.1, 0.5],
 'min_data_in_leaf': [30, 50, 100, 300, 400],
 'lambda_l1': [0, 0.3,1, 1.5],
 'lambda_l2': [0,0.3, 1],
 'n_estimators': [100,500,1000,1500],
 'n_jobs':[-1]
 }
lb = LGBMClassifier()
lb_model = RandomizedSearchCV(lb, params, cv = 3,verbose=1,n_jobs=-1)
lb_model.fit(X_train_features, train_labels,verbose=0)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent wo
rkers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:    7.7s finishe
d
```

Out[28]:

```
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=LGBMClassifier(boosting_type='gbdt',
                                            class_weight=None,
                                            colsample_bytree=1.0,
                                            importance_type='split',
                                            learning_rate=0.1, max_de
pth=-1,
                                            min_child_samples=20,
                                            min_child_weight=0.001,
                                            min_split_gain=0.0,
                                            n_estimators=100, n_jobs=
-1,
                                            num_leaves=31, objective=
None,
                                            random_state=None, reg_al
pha=0.0,
                                            reg_lambda=0.0, sile...
                                            subsample_freq=0),
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'lambda_l1': [0, 0.3, 1, 1.
5],
                                        'lambda_l2': [0, 0.3, 1],
                                        'min_data_in_leaf': [30, 50,
100, 300,
                                                             400],
                                        'n_estimators': [100, 500, 10
00, 1500],
                                        'n_jobs': [-1], 'num_leaves':
[31, 127],
                                        'reg_alpha': [0.1, 0.5]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=
True,
                   return_train_score=False, scoring=None, verbose=1)
```

In [29]:

```
lb_model.best_estimator_
```

Out[29]:

```
LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_byt
ree=1.0,
               importance_type='split', lambda_l1=0.3, lambda_l2=0.3,
               learning_rate=0.1, max_depth=-1, min_child_samples=20,
               min_child_weight=0.001, min_data_in_leaf=30, min_split
_gain=0.0,
               n_estimators=1000, n_jobs=-1, num_leaves=31, objective
=None,
               random_state=None, reg_alpha=0.1, reg_lambda=0.0, sile
nt=True,
               subsample=1.0, subsample_for_bin=200000, subsample_fre
q=0)
```

In [33]:

```
# split into train and CV data
n_splits = 5
splits = list(StratifiedKFold(n_splits=n_splits, shuffle=True).split(X_train_featur

models = []
scores = np.zeros(n_splits)

print('Training...')
print('MCC training & cv') # MCC = Matthews Correlation Coefficient
for i, (idx_train, idx_cv) in enumerate(splits):

    #train and cv split
    X_train = X_train_features[idx_train, :]
    y_train = train_labels[idx_train]

    X_cv = X_train_features[idx_cv, :]
    y_cv = train_labels[idx_cv]

    #initalizing and fitting the model
    learning_rate = 0.006
    model = LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytre
                importance_type='split', lambda_l1=0.3, lambda_l2=0.3,
                learning_rate=0.006, max_depth=-1, min_child_samples=20,
                min_child_weight=0.001, min_data_in_leaf=30, min_split_gain=0.0,
                n_estimators=1000, n_jobs=-1, num_leaves=31, objective=None,
                random_state=None, reg_alpha=0.1, reg_lambda=0.0, silent=True,
                subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
    model.fit(X_train, y_train.astype(float))

    #prediction
    y_predict_train = model.predict(X_train)
    y_predict_cv = model.predict(X_cv)

    #calculating mcc metric
    score_train = matthews_corrcoef(y_train, y_predict_train)
    score_cv = matthews_corrcoef(y_cv, y_predict_cv)

    #storing the models
    models.append(model)
    scores[i] = score_cv

    #printing the train and cross validation mcc score
    print('%d %.3f %.3f' % (i, score_train, score_cv))

#average of all the scores
print('CV scores %.3f ± %.3f' % (np.mean(scores), np.std(scores)))
```

```
Training...
MCC training & cv
0 1.000 0.795
1 1.000 0.691
2 1.000 0.623
3 1.000 0.664
4 1.000 0.795
CV scores 0.714 ± 0.070
```

## Catboost Classifier

In [24]:

```
params = {'depth': [5,6,7,8,9],
 'learning_rate' : [0.0001, 0.001,0.005, 0.01, 0.1],
 'l2_leaf_reg': [2,4,6,8],
 'iterations': [100,200,300,400]}
cb = CatBoostClassifier()
cb_model = RandomizedSearchCV(cb, params,verbose=1,cv=3,n_jobs=-1)
cb_model.fit(X_train_features, train_labels.astype(float),verbose=0)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent wo
rkers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  2.2min finishe
d
```

Out[24]:

```
RandomizedSearchCV(cv=3, error_score=nan,
                   estimator=<catboost.core.CatBoostClassifier object
at 0x7f2e605b5d30>,
                   iid='deprecated', n_iter=10, n_jobs=-1,
                   param_distributions={'depth': [5, 6, 7, 8, 9],
                                        'iterations': [100, 200, 300,
400],
                                        'l2_leaf_reg': [2, 4, 6, 8],
                                        'learning_rate': [0.0001, 0.0
01, 0.005,
                                                          0.01, 0.
1]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=
True,
                   return_train_score=False, scoring=None, verbose=1)
```

In [25]:

```
cb_model.best_estimator_
```

Out[25]:

```
<catboost.core.CatBoostClassifier at 0x7f2e62ce7c18>
```

In [26]:

```
cb_model.best_params_
```

Out[26]:

```
{'depth': 9, 'iterations': 400, 'l2_leaf_reg': 4, 'learning_rate': 0.
01}
```

In [27]:

```
# split into train and CV data
n_splits = 5
splits = list(StratifiedKFold(n_splits=n_splits, shuffle=True).split(X_train_featur

models = []
scores = np.zeros(n_splits)

print('Training...')
print('MCC training & cv') # MCC = Matthews Correlation Coefficient
for i, (idx_train, idx_cv) in enumerate(splits):

    #train and cv split
    X_train = X_train_features[idx_train, :]
    y_train = train_labels[idx_train]

    X_cv = X_train_features[idx_cv, :]
    y_cv = train_labels[idx_cv]

    #initalizing and fitting the model
    learning_rate = 0.006
    model = CatBoostClassifier(learning_rate=0.01,depth=9,l2_leaf_reg=4, od_type='I
                               loss_function='Logloss', use_best_model=True, eval_metr
    model.fit(X_train, y_train.astype(float), eval_set=(X_cv, y_cv.astype(float)),s

    #prediction
    y_predict_train = model.predict(X_train)
    y_predict_cv = model.predict(X_cv)

    #calculating mcc metric
    score_train = matthews_corrcoef(y_train, y_predict_train)
    score_cv = matthews_corrcoef(y_cv, y_predict_cv)

    #storing the models
    models.append(model)
    scores[i] = score_cv

    #printing the train and cross validation mcc score
    print('%d %.3f %.3f' % (i, score_train, score_cv))

#average of all the scores
print('CV scores %.3f ± %.3f' % (np.mean(scores), np.std(scores)))
```

```
Training...
MCC training & cv
0 0.829 0.787
1 0.607 0.632
2 0.820 0.769
3 0.712 0.739
4 0.926 0.695
CV scores 0.724 ± 0.056
```

catboostclassifier got the best mcc score

In [ ]:

```
#saving the model
'''
for i,model in enumerate(models):
    filename = 'model'+str(i)+'.sav'
    pickle.dump(model,open(filename,'wb'))
'''
```

## Evaluating the catboost model on test dataset

In [10]:

```
test_spectra_features = np.load('testfeatures/test_spectra_features.npy')
```

In [11]:

```
test_spectra_features.shape
```

Out[11]:

```
(6779, 57)
```

In [12]:

```
test_signal_features = np.load('testfeatures/test_signal_features.npy')
```

In [13]:

```
test_signal_features.shape
```

Out[13]:

```
(6779, 19)
```

In [14]:

```
test_features = np.concatenate((test_spectra_features,test_signal_features),axis=1)
```

In [15]:

```
test_features.shape
```

Out[15]:

```
(6779, 76)
```

#### Prediction on test dataset

In [19]:

```
y_test_probas = np.empty((test_features.shape[0], n_splits))

for i, model in enumerate(models):
    y_test_probas[:, i] = model.predict_proba(test_features)[:, 1]

#taking mean of all the predicted
y_test_proba = np.mean(y_test_probas, axis=1)

# Converting to 0 1 with a threshold 0.25, then replicating 3 copies for 3 phases
y_submit = np.repeat(y_test_proba > 0.25, 3)

print('Positive fraction %d/%d = %.3f' % (
    np.sum(y_submit), len(y_submit), np.sum(y_submit)/len(y_submit)))
```

```
Positive fraction 828/20337 = 0.041
```

Creating a dataframe and converting it into csv for submission

In [9]:

```
results_df = pd.DataFrame()
```

In [10]:

```
signal_id = list(range(len(y_submit)))
signal_id = [i+8712 for i in signal_id]
```

In [11]:

```
results_df['signal_id'] = signal_id
results_df['target'] = y_submit
```

In [12]:

```
results_df
```

Out[12]:

| | signal_id | target |
|---|---|---|
| **0** | 8712 | False |
| **1** | 8713 | False |
| **2** | 8714 | False |
| **3** | 8715 | False |
| **4** | 8716 | False |
| **...** | ... | ... |
| **20332** | 29044 | False |
| **20333** | 29045 | False |
| **20334** | 29046 | False |
| **20335** | 29047 | False |
| **20336** | 29048 | False |

20337 rows × 2 columns

In [13]:

```
results_df.to_csv('submission1.csv',index=False)
```

In [18]:

```
#using the above results(mcc score of 62) to increase the data points
knowledge_data = pd.read_csv('submission1.csv')
```

In [19]:

```
len_train=len(X_train_features)

y_list=knowledge_data['target'].values
y_test=[]
for j in range(0,len(y_list),3):
    y_test.append(y_list[j])
y_test=np.asarray(y_test)
del knowledge_data

print(X_train_features.shape,train_labels.shape)
print(test_features.shape,y_test.shape)

X = np.concatenate([X_train_features,test_features])
Y = np.concatenate([train_labels,y_test])
```

```
(2904, 76) (2904,)
(6779, 76) (6779,)
```

In [20]:

```
# split into train and CV data
n_splits = 5

models = []
scores = np.zeros(n_splits)

print('Training...')
print('MCC training & cv') # MCC = Matthews Correlation Coefficient


seeds=[0,42,1204,2019]
for seed in seeds:
    splits = list(StratifiedKFold(n_splits=n_splits, shuffle=True, random_state=see

    for i, (idx_train, idx_cv) in enumerate(splits):
        X_train = X[idx_train, :]
        y_train = Y[idx_train]

        X_cv = X[idx_cv, :]
        y_cv = Y[idx_cv]

        # Learning rate is important; large values overfit the data
        learning_rate = 0.006
        model = CatBoostClassifier(learning_rate=learning_rate, od_type='IncToDec',
                                   loss_function='Logloss', use_best_model=True, eval_

        model.fit(X_train, y_train.astype(float), silent=True,
                  eval_set=(X_cv, y_cv.astype(float)))

        y_predict_train = model.predict(X_train)
        y_predict_cv = model.predict(X_cv)

    #     score_train = sklearn.metrics.matthews_corrcoef(y_train, y_predict_train)
        score_cv = matthews_corrcoef(y_cv, y_predict_cv)

        models.append(model)
        scores[i] = score_cv

#     print('%d %.3f %.3f' % (i, score_train, score_cv))

print('CV scores %.3f ± %.3f' % (np.mean(scores), np.std(scores)))
```

```
Training...
MCC training & cv
CV scores 0.880 ± 0.018
```

In [21]:

```
y_test_probas = np.empty((test_features.shape[0], n_splits*len(seeds)))

# assert(len(models) == n_splits)
for i, model in enumerate(models):
    y_test_probas[:, i] = model.predict_proba(test_features)[:, 1]

y_test_proba = np.mean(y_test_probas, axis=1)

# Convert to 0 1 with a threshold 0.25, then replicate 3 copies for 3 phases
y_submit = np.repeat(y_test_proba > 0.30, 3)

print('Positive fraction %d/%d = %.3f' % (
    np.sum(y_submit), len(y_submit), np.sum(y_submit)/len(y_submit)))
```

```
Positive fraction 894/20337 = 0.044
```

In [22]:

```
results_df = pd.DataFrame()
```

In [23]:

```
signal_id = list(range(len(y_submit)))
signal_id = [i+8712 for i in signal_id]
```

In [24]:

```
results_df['signal_id'] = signal_id
results_df['target'] = y_submit
```

In [25]:

```
results_df
```

Out[25]:

| | signal_id | target |
|---|---|---|
| **0** | 8712 | False |
| **1** | 8713 | False |
| **2** | 8714 | False |
| **3** | 8715 | False |
| **4** | 8716 | False |
| **...** | ... | ... |
| **20332** | 29044 | False |
| **20333** | 29045 | False |
| **20334** | 29046 | False |
| **20335** | 29047 | False |
| **20336** | 29048 | False |

20337 rows × 2 columns

In [26]:

```
results_df.to_csv('submission2.csv',index=False)
```