Code

main.py

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

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from tqdm import tqdm
import os
import math

from utils import *

np.random.seed(42)


# load data
train_df = pd.read_csv("./dataset/algerian_fires_train.csv")
test_df = pd.read_csv("./dataset/algerian_fires_test.csv")
test_y = test_df.loc[:, 'Classes'].values
train_idx = np.arange(train_df.shape[0])
test_idx = np.arange(train_df.shape[0], train_df.shape[0] +
test_df.shape[0])
df = pd.concat([train_df, test_df], axis=0).reset_index().drop(columns=
['index'])
df.loc[test_idx, 'Classes'] = -1
df.tail(3)
df.info()


# ## **Data Process**


# feature normalization
Date_col = 'Date'
tar_col = 'Classes'
feat_cols = [col for col in train_df.columns if col != Date_col and col !=
tar_col]

df.loc[:, feat_cols] = (df.loc[:, feat_cols] - df.loc[:,
feat_cols].mean(axis=0)) / (df.loc[:, feat_cols].std(axis=0) + 1e-8)
df.head(3)


# data visualization
train_df = df.loc[train_idx, feat_cols+[tar_col]]

for col in feat_cols:
    _ = sns.displot(train_df, x=col, hue=tar_col, kde=True)
```

```
    plt.savefig("./figs/hist_{}.png".format(col), dpi=200)
    plt.show()


# features correlation
corr = df.loc[:, feat_cols+[tar_col]].corr()
_ = plt.figure(figsize=(10, 8))
_ = plt.imshow(corr)
_ = plt.colorbar()
_ = plt.xticks(np.arange(len(feat_cols+[tar_col])), feat_cols+[tar_col],
rotation=45)
_ = plt.yticks(np.arange(len(feat_cols+[tar_col])), feat_cols+[tar_col])
_ = plt.savefig("./figs/corr.png", dpi=200)
plt.show()


# time series feature consturction
df, time_cols = add_time_features(df, feat_cols)
df.head()


# ## **Logistic Regression**


from sklearn.linear_model import LogisticRegression

# train on the original dataset
log_clf = LogisticRegression()
params = {'penalty':['l1', 'l2', 'elasticnet'], 'C': [0.001, 0.01, 0.1, 1,
10], 'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']}
log_clfs, log_res_df = finetune(df.loc[train_idx, feat_cols+[tar_col]],
feat_cols, tar_col, log_clf, params)
log_res_df.to_csv("./res/ori_log_res.csv", index=False)

# predict on the test dataset
best_log_clf = log_clfs.best_estimator_
best_log_clf.fit(df.loc[train_idx, feat_cols].values, df.loc[train_idx,
tar_col].values)
log_f1, log_acc, log_cnf = get_clf_metrics(df.loc[test_idx, feat_cols+
[tar_col]], feat_cols, test_y, best_log_clf)

print("Best LOGClassifier: ", log_clfs.best_estimator_)
print("F1-score of LogisticRegression is {:.3f}".format(log_f1))
print("Accuracy of LogisticRegression is {:.3f}".format(log_acc))
print("The confusion matrix of LogisticRegression:")
plot_cnf(log_cnf, name='ori_log')


# train on the augmented dataset
log_clf = LogisticRegression()
params = {'penalty':['l1', 'l2', 'elasticnet'], 'C': [0.001, 0.01, 0.1, 1,
10], 'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']}
log_clfs, log_res_df = finetune(df.loc[train_idx, feat_cols+time_cols+
[tar_col]], feat_cols+time_cols, tar_col, log_clf, params)
log_res_df.to_csv("./res/aug_log_res.csv", index=False)

# predict on the test dataset
best_log_clf = log_clfs.best_estimator_
```

```
best_log_clf.fit(df.loc[train_idx, feat_cols+time_cols].values, df.loc[train_idx, tar_col].values)
log_f1, log_acc, log_cnf = get_clf_metrics(df.loc[test_idx, feat_cols+time_cols+[tar_col]], feat_cols+time_cols, test_y, best_log_clf)

print("Best LOGClassifier: ", log_clfs.best_estimator_)
print("F1-score of LogisticRegression is {:.3f}".format(log_f1))
print("Accuracy of LogisticRegression is {:.3f}".format(log_acc))
print("The confusion matrix of LogisticRegression:")
plot_cnf(log_cnf, name='aug_log')


# ## **Support Vector Machine**


from sklearn.svm import SVC

# train on the original dataset
svm_clf = SVC()
params = {'kernel':['linear', 'poly', 'rbf', 'sigmoid'], 'C': [0.001, 0.01, 0.1, 1, 10], 'gamma': ['scale', 'auto']}
svm_clfs, svm_res_df = finetune(df.loc[train_idx, feat_cols+[tar_col]], feat_cols, tar_col, svm_clf, params)
svm_res_df.to_csv("./res/ori_svm_res.csv", index=False)

# predict on the test dataset
best_svm_clf = svm_clfs.best_estimator_
best_svm_clf.fit(df.loc[train_idx, feat_cols].values, df.loc[train_idx, tar_col].values)
svm_f1, svm_acc, svm_cnf = get_clf_metrics(df.loc[test_idx, feat_cols+[tar_col]], feat_cols, test_y, best_svm_clf)

print("Best SVMClassifier: ", svm_clfs.best_estimator_)
print("F1-score of SupportVectorMachine is {:.3f}".format(svm_f1))
print("Accuracy of SupportVectorMachine is {:.3f}".format(svm_acc))
print("The confusion matrix of SupportVectorMachine:")
plot_cnf(svm_cnf, name='ori_svm')


# train on the augmented dataset
svm_clf = SVC()
params = {'kernel':['linear', 'poly', 'rbf', 'sigmoid'], 'C': [0.001, 0.01, 0.1, 1, 10], 'gamma': ['scale', 'auto']}
svm_clfs, svm_res_df = finetune(df.loc[train_idx, feat_cols+time_cols+[tar_col]], feat_cols+time_cols, tar_col, svm_clf, params)
svm_res_df.to_csv("./res/aug_svm_res.csv", index=False)

# predict on the test dataset
best_svm_clf = svm_clfs.best_estimator_
best_svm_clf.fit(df.loc[train_idx, feat_cols+time_cols].values, df.loc[train_idx, tar_col].values)
svm_f1, svm_acc, svm_cnf = get_clf_metrics(df.loc[test_idx, feat_cols+time_cols+[tar_col]], feat_cols+time_cols, test_y, best_svm_clf)

print("Best SVMClassifier: ", svm_clfs.best_estimator_)
print("F1-score of SupportVectorMachine is {:.3f}".format(svm_f1))
print("Accuracy of SupportVectorMachine is {:.3f}".format(svm_acc))
print("The confusion matrix of SupportVectorMachine:")
```

```
plot_cnf(svm_cnf, name='aug_svm')


# ## **Neural Networks**


from sklearn.neural_network import MLPClassifier

# train on the original dataset
mlp_clf = MLPClassifier()
params = {'activation':['tanh', 'relu'], 'solver':['lbfgs', 'sgd', 'adam'], 'learning_rate':['constant', 'invscaling', 'adaptive'], 'alpha':[0.001, 0.01, 0.1, 0.5]}
mlp_clfs, mlp_res_df = finetune(df.loc[train_idx, feat_cols+[tar_col]], feat_cols, tar_col, mlp_clf, params)
mlp_res_df.to_csv("./res/ori_mlp_res.csv", index=False)

# predict on the test dataset
best_mlp_clf = mlp_clfs.best_estimator_
best_mlp_clf.fit(df.loc[train_idx, feat_cols].values, df.loc[train_idx, tar_col].values)
mlp_f1, mlp_acc, mlp_cnf = get_clf_metrics(df.loc[test_idx, feat_cols+[tar_col]], feat_cols, test_y, best_mlp_clf)

print("Best MLPClassifier: ", mlp_clfs.best_estimator_)
print("F1-score of NeuralNetwork is {:.3f}".format(mlp_f1))
print("Accuracy of NeuralNetwork is {:.3f}".format(mlp_acc))
print("The confusion matrix of NeuralNetwork:")
plot_cnf(mlp_cnf, name='ori_mlp')


# train on the augmented dataset
mlp_clf = MLPClassifier()
params = {'activation':['tanh', 'relu'], 'solver':['lbfgs', 'sgd', 'adam'], 'learning_rate':['constant', 'invscaling', 'adaptive'], 'alpha':[0.001, 0.01, 0.1, 0.5]}
mlp_clfs, mlp_res_df = finetune(df.loc[train_idx, feat_cols+time_cols+[tar_col]], feat_cols+time_cols, tar_col, mlp_clf, params)
mlp_res_df.to_csv("./res/aug_mlp_res.csv", index=False)

# predict on the test dataset
best_mlp_clf = mlp_clfs.best_estimator_
best_mlp_clf.fit(df.loc[train_idx, feat_cols+time_cols].values, df.loc[train_idx, tar_col].values)
mlp_f1, mlp_acc, mlp_cnf = get_clf_metrics(df.loc[test_idx, feat_cols+time_cols+[tar_col]], feat_cols+time_cols, test_y, best_mlp_clf)

print("Best MLPClassifier: ", mlp_clfs.best_estimator_)
print("F1-score of NeuralNetwork is {:.3f}".format(mlp_f1))
print("Accuracy of NeuralNetwork is {:.3f}".format(mlp_acc))
print("The confusion matrix of NeuralNetwork:")
plot_cnf(mlp_cnf, name='aug_mlp')


# ## **Trivial System**


from model import TrivialClassifier
```

```
194
195 # train on the original dataset
196 trv_clf = TrivialClassifier()
197 mean_val, std_val = cross_val(df.loc[train_idx, feat_cols+[tar_col]],
    feat_cols, tar_col, trv_clf)
198 print("Mean accuracy on validation dataset: {:.3f}".format(mean_val))
199 print("Std accuracy on validation dataset: {:.3f}".format(std_val))
200
201 # predict on the test dataset
202
203 trv_acc = trv_clf.score(df.loc[test_idx, feat_cols].values, test_y)
204 y_pred = trv_clf.predict(df.loc[test_idx, feat_cols].values)
205 trv_f1 = f1_score(test_y, y_pred, average='macro')
206 trv_cnf = confusion_matrix(test_y, y_pred)
207 print("F1-score of TrivialClassifier is {:.3f}".format(trv_f1))
208 print("Accuracy of TrivialClassifier is {:.3f}".format(trv_acc))
209 print("The confusion matrix of TrivialClassifier:")
210 plot_cnf(trv_cnf, name='ori_trv')
211
212
213 # ## **Baseline: Nearest Mean Classifier**
214
215
216 from model import KernelNearestMeansClassifier
217
218 # train on the original dataset
219 knm_clf = KernelNearestMeansClassifier()
220 mean_val, std_val = cross_val(df.loc[train_idx, feat_cols+[tar_col]],
    feat_cols, tar_col, knm_clf)
221 print("Mean accuracy on validation dataset: {:.3f}".format(mean_val))
222 print("Std accuracy on validation dataset: {:.3f}".format(std_val))
223
224 # predict on the test dataset
225
226 knm_acc = knm_clf.score(df.loc[test_idx, feat_cols].values, test_y)
227 y_pred = knm_clf.predict(df.loc[test_idx, feat_cols].values)
228 knm_f1 = f1_score(test_y, y_pred, average='macro')
229 knm_cnf = confusion_matrix(test_y, y_pred)
230 print("F1-score of KernelNearestMeansClassifier is {:.3f}".format(knm_f1))
231 print("Accuracy of KernelNearestMeansClassifier is {:.3f}".format(knm_acc))
232 print("The confusion matrix of KernelNearestMeansClassifier:")
233 plot_cnf(knm_cnf, name='ori_knm')
234
235
236 # train on the augmented dataset
237 knm_clf = KernelNearestMeansClassifier()
238 mean_val, std_val = cross_val(df.loc[train_idx, feat_cols+time_cols+
    [tar_col]], feat_cols+time_cols, tar_col, knm_clf)
239 print("Mean accuracy on validation dataset: {:.3f}".format(mean_val))
240 print("Std accuracy on validation dataset: {:.3f}".format(std_val))
241
242 # predict on the test dataset
243
244 knm_acc = knm_clf.score(df.loc[test_idx, feat_cols+time_cols].values,
    test_y)
245 y_pred = knm_clf.predict(df.loc[test_idx, feat_cols+time_cols].values)
246 knm_f1 = f1_score(test_y, y_pred, average='macro')
247 knm_cnf = confusion_matrix(test_y, y_pred)
```

```
print("F1-score of KernelNearestMeansClassifier is {:.3f}".format(knm_f1))
print("Accuracy of KernelNearestMeansClassifier is {:.3f}".format(knm_acc))
print("The confusion matrix of KernelNearestMeansClassifier:")
plot_cnf(knm_cnf, name='aug_knm')


# ## **Feature Importance**


importance = best_log_clf.coef_[0]
plt.figure(figsize=(12, 6))
_ = plt.bar([x for x in range(len(importance))], importance, color='red')
_ = plt.xticks(ticks=list(range(len(importance))),
labels=feat_cols+time_cols, rotation=80)
_ = plt.xlabel("features")
_ = plt.ylabel("importance")
_ = plt.tight_layout()
_ = plt.savefig("./figs/imp_log.png", dpi=300)
_ = plt.show()

```

model.py

```
import numpy as np


class TrivialClassifier(object):
    def __init__(self, random_state=42):
        np.random.seed(random_state)

    def fit(self, X, y):
        classes = np.unique(y)
        self.classes = classes
        prob = np.zeros(len(classes))
        for i, c in enumerate(classes):
            prob[i] = np.sum(np.where(y==c, 1, 0))
        self.prob = prob / np.sum(prob)

    def predict(self, X):
        y_pred = np.random.choice(self.classes, size=X.shape[0],
p=self.prob)
        return y_pred

    def score(self, X, y):
        y_pred = self.predict(X)
        return np.mean(y_pred.ravel() == y.ravel())

class KernelNearestMeansClassifier(object):
    def __init__(self, gamma=0.01, kernel_type='linear'):
        self.gamma = gamma
        self.kernel_type = kernel_type

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y
```

```
32
33
34      def predict(self, X):
35          X1 = self.X_train[self.y_train==0]
36          X2 = self.X_train[self.y_train==1]
37          if self.kernel_type == 'rbf':
38              dist1 = np.zeros((X1.shape[0], X1.shape[0]))
39              for i in range(X1.shape[0]):
40                  dist1[i, :] = np.sum((X1[i] - X1)**2, axis=1)
41              self.K1 = np.mean(np.exp(-self.gamma * dist1))
42
43              dist2 = np.zeros((X2.shape[0], X2.shape[0]))
44              for i in range(X2.shape[0]):
45                  dist2[i, :] = np.sum((X2[i] - X2)**2, axis=1)
46              self.K2 = np.mean(np.exp(-self.gamma * dist2))
47
48              res = np.zeros((X.shape[0], 2))
49              for i in range(X.shape[0]):
50                  res[i, 0] = -self.K1 + 2 * np.mean(np.exp(-self.gamma *
    np.sum((X[i] - X1)**2, axis=1)))
51                  res[i, 1] = -self.K2 + 2 * np.mean(np.exp(-self.gamma *
    np.sum((X[i] - X2)**2, axis=1)))
52          elif self.kernel_type == "linear":
53              self.K1 = np.mean(X1 @ X1.T)
54              self.K2 = np.mean(X2 @ X2.T)
55
56              res = np.zeros((X.shape[0], 2))
57              for i in range(X.shape[0]):
58                  res[i, 0] = -self.K1 + 2 * np.mean(X[i] @ X1.T)
59                  res[i, 1] = -self.K2 + 2 * np.mean(X[i] @ X2.T)
60          else:
61              raise NotImplementedError
62          y_pred = np.argmax(res, axis=1)
63          return y_pred
64
65      def score(self, X, y):
66          y_pred = self.predict(X)
67          acc = np.mean(y_pred==y)
68          return acc
```

utils.py

```
1   import numpy as np
2   import pandas as pd
3   from sklearn.model_selection import GridSearchCV, StratifiedKFold
4   from sklearn.metrics import f1_score, confusion_matrix
5   import matplotlib.pyplot as plt
6
7   def add_time_features(df, cols, duration=5):
8       '''
9       Add time series features to the dataframe
10      '''
11      aug_cols=[]
12      for col in cols:
13          max_col = '{}_{}max'.format(col, duration)
```

```
        min_col = '{}_{}min'.format(col, duration)
        mean_col = '{}_{}mean'.format(col, duration)
        aug_cols.append(max_col)
        aug_cols.append(min_col)
        aug_cols.append(mean_col)
        df[max_col] = 0
        df[min_col] = 0
        df[mean_col] = 0
        for i in range(duration, df.shape[0]):
            df.loc[i, max_col] = df.loc[i-duration:i, col].max()
            df.loc[i, min_col] = df.loc[i-duration:i, col].min()
            df.loc[i, mean_col] = df.loc[i-duration:i, col].mean()
    for col in aug_cols:
        df.loc[df[col]==0, col] = df[col].mean()
    return df, aug_cols

def cross_val(train_df, cols, tar_col, classifier, seed=42):
    '''
    Cross validation on the training data
    returns:
        accuracy mean and std
    '''
    X, y = train_df[cols].values, train_df[tar_col].values
    skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=seed)
    acc = []
    for tr_idx, val_idx in skf.split(X, y):
        classifier.fit(X[tr_idx], y[tr_idx])
        acc.append(classifier.score(X[val_idx], y[val_idx]))
    return np.mean(acc), np.std(acc)

def finetune(train_df, cols, tar_col, classifier, params, seed=42):
    '''
    Model selection on the training data
    returns:
        the best model and validation results
    '''
    X, y = train_df[cols].values, train_df[tar_col].values
    clfs = GridSearchCV(classifier, params)
    clfs.fit(X, y)
    # return results dataframe
    results = pd.DataFrame()
    idx = np.where(np.isnan(clfs.cv_results_['std_test_score'])==False)[0]
    results['params'] = np.array(list(map(str, clfs.cv_results_['params'])))
[idx]
    results['mean_val_score'] = clfs.cv_results_['mean_test_score'][idx]
    results['std_val_score'] = clfs.cv_results_['std_test_score'][idx]
    return clfs, results

def get_clf_metrics(test_df, cols, test_y, classifier):
    '''
    Performance of classifier on the test dataset
    returns:
        f1-score, accuracy, confusion_matrix
    '''
    X, y = test_df[cols].values, test_y
    accuracy = classifier.score(X, y)
    y_pred = classifier.predict(X)
    f1 = f1_score(y, y_pred, average='macro')
```

```
    conf_matrix = confusion_matrix(y, y_pred)
    return f1, accuracy, conf_matrix

def plot_cnf(cnf, name):
    plt.imshow(cnf)
    for i in range(2):
        for j in range(2):
            plt.text(i, j, str(cnf[i, j]), color='red')
    plt.xticks(np.arange(2), ['Positive', 'Negative'])
    plt.yticks(np.arange(2), ['True', 'False'])
    plt.savefig('./figs/cnf_{}.png'.format(name), dpi=200)
    plt.show()

```