```
import math
import os
import pickle

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from sklearn import svm
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.feature_selection import RFECV, SelectFromModel
from sklearn.linear_model import LassoCV, LinearRegression, Ridge
from sklearn.metrics import mean_absolute_error, r2_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import PolynomialFeatures
from sklearn.svm import SVR


################################################################################
### parameter

standard = True
filter_outlier = True
select_feat = False
pca = True
num_fold = 5
prefix = 'SVR'
x_label = 'Kernel (K) and Regularization parameter (C)'
save_dir = 'log/SVR'

# Linear Regression
# model_type = 'LinearRegression'

# SVR
model_type = 'SVR'
model_sele_param_kernel = ['rbf', 'linear']

# Lasso
# model_type = 'Lasso'

# Ridge
# model_type = 'Ridge'
# model_sele_param_alpha = [math.exp(i-20) for i in range(24)]

# Perceptron
# model_type = 'Perceptron'

# RBF
# model_type = 'RBF'
# model_sele_param_hidden_size = [30, 50, 100, 150, 200]


################################################################################
### Dataset Definition

def splitData(train_data):
    feature_train = train_data[:,:-1]
    label_train = train_data[:, -1, None] # keep dim
    return feature_train, label_train


def preprocess(fil_tr_feature,
               fil_tr_label,
               largeVar_set,
               tr_feature_label):
    mean_train, std_train = standardize(fil_tr_feature)
    binary_set = onehot_feature()
    ## Reduce Large Variacne Data
    for i in range(len(tr_feature_label)):
        label = (tr_feature_label[i])[0]
        if label in largeVar_set:
            for j in range(len(fil_tr_label)):
                fil_tr_feature[j][i] = math.log(2+fil_tr_feature[j][i])
```

```
    ## Sandardize Data Except One-Hot Features
    for i in range(len(tr_feature_label)):
        label = (tr_feature_label[i])[0]
        if label in binary_set:
            continue
        avg = mean_train[i]
        std = std_train[i]
        for j in range(len(fil_tr_label)):
            fil_tr_feature[j][i] = (fil_tr_feature[j][i]-avg)/std

    return fil_tr_feature


def standardize(X_train):
        scaler = StandardScaler()
        scaler.fit(X_train)

        #Standardized Data
        X_train_standard = scaler.transform(X_train)

        #compute std and mean
        mean_train = scaler.mean_
        X1_train = X_train[0]
        X1_train_std = X_train_standard[0]
        std_train = (X1_train-mean_train)/X1_train_std

        return mean_train, std_train


def onehot_feature():
        binary_set = set()
        binary_set.add('n_non_stop_words');
        binary_set.add('data_channel_is_lifestyle')
        binary_set.add('data_channel_is_entertainment');
        binary_set.add('data_channel_is_bus');
        binary_set.add('data_channel_is_socmed');
        binary_set.add('data_channel_is_tech');
        binary_set.add('data_channel_is_world');
        binary_set.add('weekday_is_monday');
        binary_set.add('weekday_is_tuesday');
        binary_set.add('weekday_is_wednesday');
        binary_set.add('weekday_is_thursday');
        binary_set.add('weekday_is_friday');
        binary_set.add('weekday_is_saturday');
        binary_set.add('weekday_is_sunday');
        binary_set.add('is_weekend');
        return binary_set


class Dataset(object):
    def __init__(self, feat_dir, label_dir=None):
        self.feat_dir = feat_dir
        if label_dir is not None:
            self.label_dir = label_dir


    def read_feature(self, file_dir):
        # read feature data
        with open(file_dir,'r')as file:
            read_file = csv.reader(file)
            data = []

            for ele in read_file:
                data.append(ele)

        feature_data = [] ; feature_label = []
        for item in data[1:-1]:
            feature_data.append(np.array([float(i) for i in item]))
        feature_data = np.vstack(feature_data)
        for item in data[0]:
            feature_label.append(item)
        feature_label = np.vstack(feature_label)
```

```
        return feature_data, feature_label


    def read_label(self, file_dir):
        # read label data
        with open(file_dir,'r')as file:
            read_file = csv.reader(file)
            data = []

            for ele in read_file:
                data.append(ele)

        label_data = []
        for item in data[1:-1]:
            label_data.append(np.array([float(i) for i in item]))
        label_data = np.vstack(label_data)
        return label_data


    def normlize_large_variance_feat(self, feat, lab, feat_lab):
        mean_train, std_train = standardize(feat)
        idx = 0;  idx_set = []
        for i in std_train:
            if i > 1000:
                idx_set.append(idx)
            idx += 1
        print("The features that have large variance is: ")
        largeVar_set = set()
        for i in idx_set:
            largeVar_set.add((feat_lab[i])[0])
        print(largeVar_set)
        print()
        binary_set = onehot_feature()
        print("The features that is one-hot is: ")
        print(binary_set)

        norm_feat = preprocess(feat, lab, largeVar_set, feat_lab)
        return norm_feat


class TrainDataset(Dataset):
    def __init__(self,
                 feat_dir,
                 label_dir,
                 standard=False,
                 filter_outlier=False,
                 PCA=False):

        super(TrainDataset, self).__init__(feat_dir, label_dir)

        self.raw_feat, self.feat_lab = self.read_feature(self.feat_dir)
        self.raw_lab = self.read_label(self.label_dir)
        self.raw_data = np.concatenate((self.raw_feat, self.raw_lab), axis=1)

        self.feat = self.raw_feat
        self.lab = self.raw_lab
        self.data = self.raw_data

        if filter_outlier:
            filtered_data = self.filter_outlier(self.data, self.lab)
            _, self.feat, self.lab = filtered_data

        if standard:
            self.feat = self.normlize_large_variance_feat(
                self.feat, self.lab, self.feat_lab)

        poly = PolynomialFeatures(1)
        self.feat = poly.fit_transform(self.feat)
        self.data = np.concatenate((self.feat, self.lab), axis=1)
        self.lab = np.squeeze(self.lab)


    def filter_outlier(self, data, lab):
```

```
        lab_std = np.std(lab)
        lab_mean = np.mean(lab)
        filter_data = []

        for item in data:
            if item[-1] < lab_mean + 2*lab_std:
                filter_data.append(item)

        fil_data = np.vstack(filter_data)
        fil_feat, fil_lab = splitData(fil_data)
        return fil_data, fil_feat, fil_lab


class TestDataset(Dataset):
    def __init__(self,
                 feat_dir,
                 label_dir,
                 standard=False,
                 PCA=False):

        super(TestDataset, self).__init__(feat_dir, label_dir)

        self.raw_feat, self.feat_lab = self.read_feature(self.feat_dir)

        self.feat = self.raw_feat
        if standard:
            self.feat = self.normlize_large_variance_feat(
                self.feat, self.lab, self.feat_lab)

        poly = PolynomialFeatures(1)
        self.feat = poly.fit_transform(self.feat)
        self.lab = np.squeeze(self.lab)


################################################################################
### Evaluation Metrics

"""
pred: the predicted value, shape: (N, D)
gt_lab: the ground truth, shape: (N, D)
N: number of samples
D: number of candidate models
"""

def mean_absolute_error(pred, gt_lab):
    assert pred.shape == gt_lab.shape
    N = pred.shape[0]
    mae = np.sum(np.abs(pred - gt_lab), axis=0) / N
    return mae


def r_squared(pred, gt_lab):
    assert pred.shape == gt_lab.shape
    N = pred.shape[0]
    y_mean = np.ones_like(gt_lab) * np.mean(gt_lab)
    r_squared = 1 - np.sum((gt_lab-pred) ** 2, axis=0) / np.sum((gt_lab-y_mean) ** 2
, axis=0)
    return r_squared


def pMSE(pred, gt_lab, r=10):
    assert pred.shape == gt_lab.shape
    N = pred.shape[0]
    tmp = (gt_lab - pred) / (r + gt_lab)
    pmse = np.sum(tmp ** 2, axis=0) / N
    return pmse


def pMAE(pred, gt_lab, r=10):
    assert pred.shape == gt_lab.shape
    N = pred.shape[0]
    tmp = (gt_lab - pred) / (r + gt_lab)
    pmae = np.sum(np.abs(tmp), axis=0) / N
```

```
    return pmae


def m_r_squared(pred, gt_lab, r=10):
    assert pred.shape == gt_lab.shape
    N = pred.shape[0]
    y_mean = np.ones_like(gt_lab) * np.mean(gt_lab)
    m_r_squared = 1 - pMSE(pred, gt_lab, r=r) / pMSE(y_mean, gt_lab, r=r)
    return m_r_squared


################################################################################
### Models
class RBFModule(object):
    def __init__(self, hidden_shape, centers=None, gamma=1.0):
        self.hidden_shape = hidden_shape
        self.gamma = gamma
        self.centers = centers
        self.lr = LinearRegression()
        self.poly = PolynomialFeatures(1)

    def _kernel_function(self, center, data_point):
        return np.exp(-self.gamma*np.linalg.norm(center-data_point)**2)

    def _calculate_interpolation_matrix(self, X):
        G = np.zeros((len(X), self.hidden_shape))
        for data_point_arg, data_point in enumerate(X):
            for center_arg, center in enumerate(self.centers):
                G[data_point_arg, center_arg] = self._kernel_function(
                        center, data_point)
        return G

    def _select_centers(self, X):
        #random_args = np.random.choice(len(X), self.hidden_shape)
        #centers = X[random_args]
        kmeans = KMeans(n_clusters=self.hidden_shape, init='random', random_state=0)
.fit(X)
        centers = kmeans.cluster_centers_
        return centers

    def fit(self, X, Y):
        if self.centers is None:
            self.centers = self._select_centers(X)
        G = self._calculate_interpolation_matrix(X)
        G = self.poly.fit_transform(G)
        self.lr.fit(G, Y)
        # self.weights = np.dot(np.linalg.pinv(G), Y)

    def predict(self, X):
        G = self._calculate_interpolation_matrix(X)
        G = self.poly.fit_transform(G)
        predictions = self.lr.predict(G)
        # predictions = np.dot(G, self.weights)
        return predictions

################################################################################
### Visualization

# from https://matplotlib.org/stable/gallery/images_contours_and_fields/image_annota
ted_heatmap.html?highlight=heatmap
def heatmap(data, row_labels, col_labels, ax=None,
            cbar_kw={}, cbarlabel="", **kwargs):
    """
    Create a heatmap from a numpy array and two lists of labels.

    Parameters
    ----------
    data
        A 2D numpy array of shape (N, M).
    row_labels
        A list or array of length N with the labels for the rows.
    col_labels
        A list or array of length M with the labels for the columns.
```

```
    ax
        A `matplotlib.axes.Axes` instance to which the heatmap is plotted.  If
        not provided, use current axes or create a new one.  Optional.
    cbar_kw
        A dictionary with arguments to `matplotlib.Figure.colorbar`.  Optional.
    cbarlabel
        The label for the colorbar.  Optional.
    **kwargs
        All other arguments are forwarded to `imshow`.
    """

    if not ax:
        ax = plt.gca()

    # Plot the heatmap
    im = ax.imshow(data, **kwargs)

    # Create colorbar
    cbar = ax.figure.colorbar(im, ax=ax, **cbar_kw)
    cbar.ax.set_ylabel(cbarlabel, rotation=-90, va="bottom")

    # We want to show all ticks...
    ax.set_xticks(np.arange(data.shape[1]))
    ax.set_yticks(np.arange(data.shape[0]))
    # ... and label them with the respective list entries.
    ax.set_xticklabels(col_labels)
    ax.set_yticklabels(row_labels)

    # Let the horizontal axes labeling appear on top.
    ax.tick_params(top=True, bottom=False,
                   labeltop=True, labelbottom=False)

    # Rotate the tick labels and set their alignment.
    plt.setp(ax.get_xticklabels(), rotation=-30, ha="right",
             rotation_mode="anchor")

    # Turn spines off and create white grid.
    # TODO always error, need further check
    # ax.spines[:].set_visible(False)

    ax.set_xticks(np.arange(data.shape[1]+1)-.5, minor=True)
    ax.set_yticks(np.arange(data.shape[0]+1)-.5, minor=True)
    ax.grid(which="minor", color="w", linestyle='-', linewidth=3)
    ax.tick_params(which="minor", bottom=False, left=False)

    return im, cbar

# from https://matplotlib.org/stable/gallery/images_contours_and_fields/image_annota
ted_heatmap.html?highlight=heatmap
def annotate_heatmap(im, data=None, valfmt="{x:.2f}",
                     textcolors=("black", "white"),
                     threshold=None, **textkw):
    """
    A function to annotate a heatmap.

    Parameters
    ----------
    im
        The AxesImage to be labeled.
    data
        Data used to annotate.  If None, the image's data is used.  Optional.
    valfmt
        The format of the annotations inside the heatmap.  This should either
        use the string format method, e.g. "$ {x:.2f}", or be a
        `matplotlib.ticker.Formatter`.  Optional.
    textcolors
        A pair of colors.  The first is used for values below a threshold,
        the second for those above.  Optional.
    threshold
        Value in data units according to which the colors from textcolors are
        applied.  If None (the default) uses the middle of the colormap as
        separation.  Optional.
    **kwargs
```

```
        All other arguments are forwarded to each call to `text` used to create
        the text labels.
    """

    if not isinstance(data, (list, np.ndarray)):
        data = im.get_array()

    # Normalize the threshold to the images color range.
    if threshold is not None:
        threshold = im.norm(threshold)
    else:
        threshold = im.norm(data.max())/2.

    # Set default alignment to center, but allow it to be
    # overwritten by textkw.
    kw = dict(horizontalalignment="center",
              verticalalignment="center")
    kw.update(textkw)

    # Get the formatter in case a string is supplied
    if isinstance(valfmt, str):
        valfmt = matplotlib.ticker.StrMethodFormatter(valfmt)

    # Loop over the data and create a `Text` for each "pixel".
    # Change the text's color depending on the data.
    texts = []
    for i in range(data.shape[0]):
        for j in range(data.shape[1]):
            kw.update(color=textcolors[int(im.norm(data[i, j]) > threshold)])
            text = im.axes.text(j, i, valfmt(data[i, j], None), **kw)
            texts.append(text)

    return texts


def plt_distribution(data, bins, title, xlabel, ylabel, save_dir):
    fig, ax = plt.subplots()
    ax.hist(data, bins=bins)
    plt.gca().set(title=title,
                  xlabel=xlabel,
                  ylabel=ylabel)
    plt.savefig(save_dir)


def plt_corr_matrix(corr_mat, feat_lab, save_dir):
    fig, ax = plt.subplots(figsize=(15, 15))
    im, cbar = heatmap(corr_mat, feat_lab, feat_lab, ax=ax, cmap="YlGn")
    fig.tight_layout()
    plt.savefig(save_dir)


def plt_eval_metrics(x_data, y_data, prefix, x_label, save_dir):
    fig, ax = plt.subplots(5, 1, sharex=True, figsize=(15, 10))
    x = np.array(x_data)
    for i, item in enumerate(y_data.items()):
        k, v = item
        sub_ax = ax[i]
        sub_ax.plot(x, np.array(v), label=k)
        sub_ax.legend()
        np.save(os.path.join(save_dir, prefix + k), np.array(v))
    plt.xlabel('Parameters for model selection: {}'.format(x_label))
    # plt.ylabel('Performance')
    fig.tight_layout()
    plt.savefig(os.path.join(save_dir, prefix))



################################################################################
################################################################################


def cross_val(k, feat, lab, model):
    kf = KFold(n_splits=k)
```

```
    kf.get_n_splits(feat)
    mae_set = []; r2_set = []; pmse_set = []; pmae_set = []; mr2_set = []

    for idx_tr, idx_val in kf.split(feat):
        feat_tr, feat_val = feat[idx_tr], feat[idx_val]
        lab_tr, lab_val = lab[idx_tr], lab[idx_val]
        model.fit(feat_tr, lab_tr)
        pred_te = model.predict(feat_val)
        mae_set.append(mean_absolute_error(lab_val, pred_te))
        r2_set.append(r2_score(lab_val, pred_te))
        pmse_set.append(pMSE(pred_te, lab_val, r=10))
        pmae_set.append(pMAE(pred_te, lab_val, r=10))
        mr2_set.append(m_r_squared(pred_te, lab_val, r=10))

    return np.mean(mae_set), np.mean(r2_set), np.mean(pmse_set), np.mean(pmae_set),
np.mean(mr2_set)


def trainer(feat_tr, lab_tr, feat_te, lab_te, model):
    mae_set = []; r2_set = []; pmse_set = []; pmae_set = []; mr2_set = []
    model.fit(feat_tr, lab_tr)
    pred_te = model.predict(feat_te)
    mae_set.append(mean_absolute_error(lab_te, pred_te))
    r2_set.append(r2_score(lab_te, pred_te))
    pmse_set.append(pMSE(pred_te, lab_te, r=10))
    pmae_set.append(pMAE(pred_te, lab_te, r=10))
    mr2_set.append(m_r_squared(pred_te, lab_te, r=10))

    return np.mean(mae_set), np.mean(r2_set), np.mean(pmse_set), np.mean(pmae_set),
np.mean(mr2_set)


def main():
    ##########################################################################
    ### Database
    # Initialization
    data_tr = TrainDataset(feat_dir='data/NEWS_Training_data.csv',
                           label_dir='data/NEWS_Training_label.csv',
                           standard=standard,
                           filter_outlier=filter_outlier,)

    data_te = TrainDataset(feat_dir='data/NEWS_Test_data.csv',
                           label_dir='data/NEWS_Test_label.csv',
                           standard=standard,)

    """
    # Draw frequency histogram
    plt_distribution(data=data_tr.lab,
                     bins=300,
                     title='Frequency Histogram',
                     xlabel='Label (number of sharings)',
                     ylabel='Frequency',
                     save_dir=os.path.join('log', 'freq_his.png'))


    # Draw frequency histogram for filtered data
    plt_distribution(data=data_tr.fil_lab,
                     bins=100,
                     title='Frequency Histogram',
                     xlabel='Label (number of sharings)',
                     ylabel='Frequency',
                     save_dir=os.path.join('log', 'fil_freq_his.png'))

    # Correlation matrix and Plot
    corr_mat = np.corrcoef(data_tr.fil_norm_feat.T)
    plt_corr_matrix(corr_mat,
                    data_tr.feat_lab,
                    os.path.join('log', 'corr_mat.png'))
    """


    ##########################################################################
    ### Model Setting
```

```
    if model_type == 'RBF':
        model = []
        model_sele_param = []
        for i in model_sele_param_hidden_size:
            kmeans = KMeans(n_clusters=i, init='random', random_state=0).fit(data_tr
.feat)
            centers = kmeans.cluster_centers_
            gamma = (np.prod(np.ptp(data_tr.feat, axis=0)[1:]) / i) ** (1/58)
            gamma_list = [gamma/1024, gamma/512, gamma/256, gamma/128]
            for j in gamma_list:
                model.append(RBFModule(hidden_shape=i, centers=centers,gamma=j))
                model_sele_param.append('M:{}\n g:{:.2f}'.format(i, j))
    elif model_type == 'LinearRegression':
        model = LinearRegression()
    elif model_type == 'SVR':
        model = []
        model_sele_param = []
        for kernel in model_sele_param_kernel:
            if kernel == 'rbf':
                model.append(SVR(kernel=kernel))
                model_sele_param.append('K:{}\n'.format(kernel))
            else:
                C_list = [math.exp(i-2) for i in range(5)]
                for C in C_list:
                    model.append(SVR(kernel=kernel, C=C))
                    model_sele_param.append('K:{}\n C:{:.2f}'.format(kernel, C))
    elif model_type == 'Ridge':
        model = [Ridge(alpha=la) for la in model_sele_param_alpha]
        model_sele_param = model_sele_param_alpha
    elif model_type == 'Lasso':
        model = LassoCV()
    elif model_type == 'Trivial':
        model = None
    else:
        raise NotImplementedError


    ##########################################################################
    ### Model Selection (if necessary)

    if isinstance(model, (list, tuple)):
        # model selection
        mae_set = []; r2_set = []; pmse_set = []; pmae_set = []; mr2_set = []
        for param, sub_model in zip(model_sele_param, model):
            if select_feat:
                selector = SelectFromModel(estimator=sub_model)
                selector.fit(data_tr.feat, data_tr.lab)
                data_tr.feat_reduced = selector.transform(data_tr.feat)
                data_te.feat_reduced = selector.transform(data_te.feat)

                mae, r2, pmse, pmae, mr2 = cross_val(num_fold,
                                                     data_tr.feat_reduced,
                                                     data_tr.lab,
                                                     sub_model)
            mae, r2, pmse, pmae, mr2 = cross_val(num_fold,
                                                 data_tr.feat,
                                                 data_tr.lab,
                                                 sub_model)

            mae_set.append(mae)
            r2_set.append(r2)
            pmse_set.append(pmse)
            pmae_set.append(pmae)
            mr2_set.append(mr2)

        # save result
        eval_metr = {
            'mae': mae_set,
            'r2': r2_set,
            'pmse': pmse_set,
            'pmae': pmae_set,
            'mr2': mr2_set
        }
```

```
        plt_eval_metrics(
            x_data=model_sele_param,
            y_data=eval_metr,
            x_label=x_label,
            prefix=prefix,
            save_dir=save_dir,
        )

        # get the optimal model
        model = model[np.argmax(np.array(r2_set))]
        print('optim parameter:{}'.format(model_sele_param[np.argmax(np.array(r2_set
))]))


    ##########################################################################
    ### PCA feat selection
    # cannot used together with model selection and feat selection from model
    if pca:
        D = data_tr.feat.shape[1] - 1
        n_comp = [int(i) for i in range(int(D / 10), int(D), 10)]
        mae_set = []; r2_set = []; pmse_set = []; pmae_set = []; mr2_set = []
        for item in n_comp:
            pca_module = PCA(n_components=item)
            pca_module.fit(data_tr.feat)
            data_tr.pca_feat = pca_module.fit_transform(data_tr.feat)
            data_te.pca_feat = pca_module.fit_transform(data_te.feat)

            mae, r2, pmse, pmae, mr2 = trainer(data_tr.feat,
                                               data_tr.lab,
                                               data_te.feat,
                                               data_te.lab,
                                               model)

            mae_set.append(mae)
            r2_set.append(r2)
            pmse_set.append(pmse)
            pmae_set.append(pmae)
            mr2_set.append(mr2)

        eval_metr = {
            'mae': mae_set,
            'r2': r2_set,
            'pmse': pmse_set,
            'pmae': pmae_set,
            'mr2': mr2_set
        }

        plt_eval_metrics(
            x_data=n_comp,
            y_data=eval_metr,
            x_label='Number of components of PCA',
            prefix=prefix,
            save_dir=save_dir,
        )

        # get the optimal PCA_feat
        n_comp = n_comp[np.argmin(np.array(mae_set))]
        print('PCA: {}'.format(n_comp))


    ##########################################################################
    ### Inference and Save Result

    feat_tr = data_tr.feat
    feat_te = data_te.feat

    if select_feat:
        selector = SelectFromModel(estimator=model)
        selector.fit(data_tr.feat, data_tr.lab)
        data_tr.feat_reduced = selector.transform(data_tr.feat)
        data_te.feat_reduced = selector.transform(data_te.feat)
        feat_tr = data_tr.feat_reduced
```

```
        feat_te = data_te.feat_reduced

    if pca:
        pca_module = PCA(n_components=n_comp)
        pca_module.fit(data_tr.feat)
        data_tr.pca_feat = pca_module.fit_transform(data_tr.feat)
        data_te.pca_feat = pca_module.fit_transform(data_te.feat)
        feat_tr = data_tr.pca_feat
        feat_te = data_te.pca_feat


    model.fit(feat_tr, data_tr.lab)
    pickle.dump(model, open(os.path.join(save_dir, prefix + '.pkl'), 'wb'))
    pred_te = model.predict(feat_te)
    print('{} model measure on test set'.format(model_type))

    print('MAE: {}'.format(mean_absolute_error(data_te.lab, pred_te)))
    print('R2: {}'.format(r2_score(data_te.lab, pred_te)))
    print('pMSE: {}'.format(pMSE(pred_te, data_te.lab, r=10)))
    print('pMAE: {}'.format(pMAE(pred_te, data_te.lab, r=10)))
    print('mR2: {}'.format(m_r_squared(pred_te, data_te.lab, r=10)))



if __name__ == '__main__':
    main()
```