```python
import util
import numpy as np
import matplotlib.pyplot as plt

np.seterr(all='raise')


factor = 2.0

class LinearModel(object):
    """Base class for linear models."""

    def __init__(self, theta=None):
        """
        Args:
            theta: Weights vector for the model.
        """
        self.theta = theta

    def fit(self, X, y):
        """Run solver to fit linear model. You have to update the value of
        self.theta using the normal equations.

        Args:
            X: Training example inputs. Shape (n_examples, dim).
            y: Training example labels. Shape (n_examples,).
        """
        # *** START CODE HERE ***

        y_new = np.expand_dims(y, axis=1)
        self.theta = np.linalg.solve(np.dot(X.T, X), np.dot(X.T, y_new))

        # *** END CODE HERE ***

    def create_poly(self, k, X):
        """
        Generates a polynomial feature map using the data x.
        The polynomial map should have powers from 0 to k
        Output should be a numpy array whose shape is (n_examples, k+1)

        Args:
            X: Training example inputs. Shape (n_examples, 2).
        """
        # *** START CODE HERE ***
        for i in range(2, k+1):
            pow_x=np.power(X[:,1], i)
            pow_x=np.expand_dims(pow_x, axis=1)
            X=np.hstack([X, pow_x])
        return X
        # *** END CODE HERE ***

    def create_sin(self, k, X):
        """
        Generates a sin with polynomial featuremap to the data x.
        Output should be a numpy array whose shape is (n_examples, k+2)

        Args:
            X: Training example inputs. Shape (n_examples, 2).
        """
        # *** START CODE HERE ***
        sin_x=np.sin(X[:,1])
        sin_x=np.expand_dims(sin_x, axis=1)
        for i in range(2, k+1):
            pow_x=np.power(X[:,1], i)
            pow_x=np.expand_dims(pow_x, axis=1)
            X=np.hstack([X, pow_x])
        X=np.hstack([X,sin_x])
        return X
        # *** END CODE HERE ***

    def predict(self, X):
        """
```

```
        Make a prediction given new inputs x.
        Returns the numpy array of the predictions.

        Args:
            X: Inputs of shape (n_examples, dim).

        Returns:
            Outputs of shape (n_examples,).
        """
        # *** START CODE HERE ***
        return np.dot(X, self.theta)
        # *** END CODE HERE ***


def run_exp(train_path, sine=False, ks=[1, 2, 3, 5, 10, 20], filename='plot.png'):
    train_x,train_y=util.load_dataset(train_path,add_intercept=True)
    plot_x = np.ones([1000, 2])
    plot_x[:, 1] = np.linspace(-factor*np.pi, factor*np.pi, 1000)
    plt.figure()
    plt.scatter(train_x[:, 1], train_y)

    for k in ks:
        '''
        Our objective is to train models and perform predictions on plot_x data
        '''
        # *** START CODE HERE ***
        lmod = LinearModel()
        print(k, sine, filename)
        if sine:
            train_phi = lmod.create_sin(k, train_x)
            plot_phi = lmod.create_sin(k, plot_x)
        else:
            train_phi = lmod.create_poly(k,train_x)
            plot_phi = lmod.create_poly(k, plot_x)
        lmod.fit(train_phi, train_y)

        plot_y = lmod.predict(plot_phi)
        # *** END CODE HERE ***
        '''
        Here plot_y are the predictions of the linear model on the plot_x data
        '''
        plt.ylim(-2, 2)
        plt.plot(plot_x[:, 1], plot_y, label='k=%d' % k)

    plt.legend()
    plt.savefig(filename)
    plt.clf()


def main(train_path, small_path, eval_path):
    '''
    Run all expetriments
    '''
    # *** START CODE HERE ***
    run_exp(train_path, False, [3], 'large-poly3.png')
    run_exp(train_path, False, [3, 5, 10, 20], 'large-poly.png')
    run_exp(train_path, True, [0, 1, 2, 3, 5, 10, 20], 'large-sine.png')
    run_exp(small_path, True, [1, 2, 5, 10, 20], 'small-sine.png')
    run_exp(small_path, False, [1, 2, 5, 10, 20], 'small-poly.png')
    # *** END CODE HERE ***

if __name__ == '__main__':
    main(train_path='train.csv',
        small_path='small.csv',
        eval_path='test.csv')
```