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

def main(lr, train_path, eval_path, save_path):
    """Problem: Poisson regression with gradient ascent.

    Args:
        lr: Learning rate for gradient ascent.
        train_path: Path to CSV file containing dataset for training.
        eval_path: Path to CSV file containing dataset for evaluation.
        save_path: Path to save predictions.
    """
    # Load training set
    x_train, y_train = util.load_dataset(train_path, add_intercept=True)

    # *** START CODE HERE ***
    # Fit a Poisson Regression model
    clf = PoissonRegression(step_size=lr)
    clf.fit(x_train, y_train)

    # Run on the validation set, and use np.savetxt to save outputs to save_path
    x_eval, y_eval = util.load_dataset(eval_path, add_intercept=True)
    p_eval = clf.predict(x_eval)
    np.savetxt(save_path, p_eval)
    plt.figure()
    plt.scatter(y_eval,p_eval,alpha=0.4,c='red',label='Ground Truth vs Predicted')
    plt.xlabel('Ground Truth')
    plt.ylabel('Predictions')
    plt.legend()
    plt.savefig('poisson_valid.png')
    # *** END CODE HERE ***


class PoissonRegression:
    """Poisson Regression.

    Example usage:
        > clf = PoissonRegression(step_size=lr)
        > clf.fit(x_train, y_train)
        > clf.predict(x_eval)
    """

    def __init__(self, step_size=1e-5, max_iter=10000000, eps=1e-5,
                 theta_0=None, verbose=True):
        """
        Args:
            step_size: Step size for iterative solvers only.
            max_iter: Maximum number of iterations for the solver.
            eps: Threshold for determining convergence.
            theta_0: Initial guess for theta. If None, use the zero vector.
            verbose: Print loss values during training.
        """
        self.theta = theta_0
        self.step_size = step_size
        self.max_iter = max_iter
        self.eps = eps
        self.verbose = verbose

    def fit(self, x, y):
        """Run gradient ascent to maximize likelihood for Poisson regression.

        Args:
            x: Training example inputs. Shape (n_examples, dim).
            y: Training example labels. Shape (n_examples,).
        """
        # *** START CODE HERE ***
        m, n = x.shape
        if self.theta is None:
            self.theta = np.zeros(n, dtype=np.float32)

        prev_theta = None
        i = 0
        while i < self.max_iter \
                and (prev_theta is None
                     or np.sum(np.abs(self.theta - prev_theta)) > self.eps):
            i += 1
            prev_theta = np.copy(self.theta)
            self._step(x, y)
            if self.verbose and i % 5 == 0:
                print('[iter: {:02d}, theta: {}]'
                      .format(i, [round(t, 5) for t in self.theta]))
        # *** END CODE HERE ***

    def predict(self, x):
        """Make a prediction given inputs x.

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

        Returns:
            Floating-point prediction for each input, shape (n_examples,).
        """
        # *** START CODE HERE ***
        y_hat = np.exp(x.dot(self.theta))

        return y_hat

    def _step(self, x, y):
        """Perform a single gradient ascent update step."""
        grad = np.expand_dims(y - np.exp(x.dot(self.theta)), 1) * x
        self.theta = self.theta + self.step_size * np.sum(grad, axis=0)
        # *** END CODE HERE ***

if __name__ == '__main__':
    main(lr=1e-5,
        train_path='train.csv',
        eval_path='valid.csv',
        save_path='poisson_pred.txt')
```