```python
import numpy as np
import util


def main(train_path, valid_path, save_path):
    """Problem: Gaussian discriminant analysis (GDA)

    Args:
        train_path: Path to CSV file containing dataset for training.
        valid_path: Path to CSV file containing dataset for validation.
        save_path: Path to save predicted probabilities using np.savetxt().
    """
    # Load dataset
    x_train, y_train = util.load_dataset(train_path, add_intercept=False)

    # *** START CODE HERE ***
    # Train a GDA classifier
    clf = GDA()
    clf.fit(x_train, y_train)

    # Plot decision boundary on validation set
    x_eval, y_eval = util.load_dataset(valid_path, add_intercept=False)
    plot_path = save_path.replace('.txt', '.png')
    util.plot(x_eval, y_eval, clf.theta, plot_path)
    x_eval = util.add_intercept(x_eval)

    # Use np.savetxt to save outputs from validation set to save_path
    p_eval = clf.predict(x_eval)
    yhat = p_eval > 0.5
    print('GDA Accuracy: %.2f' % np.mean( (yhat == 1) == (y_eval == 1)))
    np.savetxt(save_path, p_eval)
    # *** END CODE HERE ***


class GDA:
    """Gaussian Discriminant Analysis.

    Example usage:
        > clf = GDA()
        > clf.fit(x_train, y_train)
        > clf.predict(x_eval)
    """
    def __init__(self, step_size=0.01, max_iter=10000, 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):
        """Fit a GDA model to training set given by x and y by updating
        self.theta.

        Args:
            x: Training example inputs. Shape (n_examples, dim).
            y: Training example labels. Shape (n_examples,).
        """
        # *** START CODE HERE ***
        m, n = x.shape

        # Find phi, mu_0, mu_1, and sigma
        phi = 1 / m * np.sum(y == 1)
        mu_0 = (y == 0).dot(x) / np.sum(y == 0)
        mu_1 = (y == 1).dot(x) / np.sum(y == 1)
        mu_yi = np.where(np.expand_dims(y == 0, -1),
                         np.expand_dims(mu_0, 0),
                         np.expand_dims(mu_1, 0))
        sigma = 1 / m * (x - mu_yi).T.dot(x - mu_yi)

        # Write theta in terms of the parameters
        self.theta = np.zeros(n + 1)
        sigma_inv = np.linalg.inv(sigma)
        mu_diff = mu_0.T.dot(sigma_inv).dot(mu_0) \
            - mu_1.T.dot(sigma_inv).dot(mu_1)
        self.theta[0] = 1 / 2 * mu_diff - np.log((1 - phi) / phi)
        self.theta[1:] = -sigma_inv.dot(mu_0 - mu_1)

        if self.verbose:
            print('Final theta (GDA): {}'.format(self.theta))
        # *** END CODE HERE ***

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

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

        Returns:
            Outputs of shape (n_examples,).
        """
        # *** START CODE HERE ***
        y_hat = self._sigmoid(x.dot(self.theta))

        return y_hat

    @staticmethod
    def _sigmoid(x):
        return 1 / (1 + np.exp(-x))
        # *** END CODE HERE

if __name__ == '__main__':
    main(train_path='ds1_train.csv',
         valid_path='ds1_valid.csv',
         save_path='gda_pred_1.txt')

    main(train_path='ds2_train.csv',
         valid_path='ds2_valid.csv',
         save_path='gda_pred_2.txt')
```