```python
import numpy as np
import util


def main(train_path, valid_path, save_path):
    """Problem: Logistic regression with Newton's Method.

    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().
    """
    x_train, y_train = util.load_dataset(train_path, add_intercept=True)

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

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

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


class LogisticRegression:
    """Logistic regression with Newton's Method as the solver.

    Example usage:
        > clf = LogisticRegression()
        > clf.fit(x_train, y_train)
        > clf.predict(x_eval)
    """
    def __init__(self, step_size=0.01, max_iter=1000000, 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 Newton's Method to minimize J(theta) for logistic 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)

        for i in range(self.max_iter):
            grad = self._gradient(x, y)
            hess = self._hessian(x)

```

vgpatel1@stanford.edu - Dec 9, 2021, 3:45:42 PM PST

```
            prev_theta = np.copy(self.theta)
            self.theta -= self.step_size * np.linalg.inv(hess).dot(grad)

            loss = self._loss(x, y)
            if self.verbose:
                print('[iter: {:02d}, loss: {:.7f}]'.format(i, loss))

            if np.sum(np.abs(prev_theta - self.theta)) < self.eps:
                break

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

    def predict(self, x):
        """Return predicted probabilities 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

    def _gradient(self, x, y):
        """Get gradient of J.

        Returns:
            grad: The gradient of J with respect to theta. Same shape as theta.
        """
        m, _ = x.shape

        probs = self._sigmoid(x.dot(self.theta))
        grad = 1 / m * x.T.dot(probs - y)

        return grad

    def _hessian(self, x):
        """Get the Hessian of J given theta and x.

        Returns:
            hess: The Hessian of J. Shape (dim, dim), where dim is dimension of theta.
        """
        m, _ = x.shape

        probs = self._sigmoid(x.dot(self.theta))
        diag = np.diag(probs * (1. - probs))
        hess = 1 / m * x.T.dot(diag).dot(x)

        return hess

    def _loss(self, x, y):
        """Get the empirical loss for logistic regression."""
        hx = self._sigmoid(x.dot(self.theta))
        loss = -np.mean(y * np.log(hx + self.eps) + (1 - y) * np.log(1 - hx + self.eps))

        return loss

    @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='logreg_pred_1.txt')
```

vgpatel1@stanford.edu - Dec 9, 2021, 3:45:42 PM PST

```
    main(train_path='ds2_train.csv',
         valid_path='ds2_valid.csv',
         save_path='logreg_pred_2.txt')
```

vgpatel1@stanford.edu - Dec 9, 2021, 3:45:42 PM PST