ML2

Pautrat Rémi (M.) <remi.pautrat@polytechnique.edu>

lun. 06/02/2017 15:33

À :Oltean Matei (M.) <matei.oltean@polytechnique.edu>;

Cc De Lavaissière De Lavergne Stéfan (M.) <stefan.de-lavergne@polytechnique.edu>; Boschin Armand (M.) <armand.boschin@polytechnique.edu>;

```
class ModelLogReg:
    """A class giving first order information for logistic regression

    Parameters
    ----------
    X : `numpy.array`, shape=(n_samples, n_features)
        The features matrix

    y : `numpy.array`, shape=(n_samples,)
        The vector of labels

    strength : `float`
        The strength of ridge penalization
    """
    def __init__(self, X, y, strength):
        self.X = X
        self.y = y
        self.strength = strength
        self.n_samples, self.n_features = X.shape

    def loss(self, w):
        """Computes f(w)"""
        y, X, n_samples, strength = self.y, self.X, self.n_samples, self.strength
        ### TODO
        res = 0
        for i in range(n_samples):
            res += np.log(1 + np.exp(-y[i]*X[i].dot(w)))
        res /= n_samples
        res += strength * norm(w) ** 2 / 2
        return res
        ### END TODO

    def grad_i(self, i, w):
        """Computes the gradient of f_i at w"""
        x_i = self.X[i]
        strength = self.strength
        ### TODO
        aux = np.exp(-y[i]*x_i.dot(w))
        return (-y[i] * aux / (1 + aux) ) * x_i + strength * w
        ### END TODO

    def grad(self, w):
        """Computes the gradient of f at w"""
        y, X, n_samples, strength = self.y, self.X, self.n_samples, self.strength
```

```
        ### TODO
        res = 0
        for i in range(n_samples):
            res += self.grad_i(i, w)
        res /= n_samples
        return res
        ### END TODO

    def grad_coordinate(self, j, w):
        """Computes the partial derivative of f with respect to
        the j-th coordinate"""
        y, X, n_samples, strength = self.y, self.X, self.n_samples, self.strength
        ### TODO
        res = 0
        for i in range(n_samples):
            aux = np.exp(-y[i] * X[i].dot(w))
            res -= y[i] * X[i][j] * aux / (1 + aux)
        res /= n_samples
        res += strength * w[j]
        return res
        ### END TODO

    def lip(self):
        """Computes the Lipschitz constant of f"""
        X, n_samples = self.X, self.n_samples
        ### TODO
        return norm(X.T.dot(X), 2) / (4 * n_samples) + self.strength
        ### END TODO

    def lip_coordinates(self):
        """Computes the Lipschitz constant of f with respect to
        the j-th coordinate"""
        X, n_samples = self.X, self.n_samples
        ### TODO
        return (X ** 2).sum(axis=0) / (4 * n_samples) + self.strength
        ### END TODO

    def lip_max(self):
        """Computes the maximum of the lipschitz constants of f_i"""
        X, n_samples = self.X, self.n_samples
        ### TODO
        return ((X ** 2).sum(axis=1) / 4 + self.strength).max()
        ### END TODO
```