Machine Learning 2016 Fall

hw2 – Spam Classification

b03902035 黃兆緯

1. Logistic Regression Function

```python
    """
    Fit training data
    params:
        X:          Numpy array with shape (n_samples, n_features)
        y:          Numpy array with shape (n_samples, 1)
    """
    def fit(self, X, y):
        if len(y.shape) == 1:
            y = np.array([[num] for num in y])

        if len(X) == 0 or len(X) != len(y):
            print('length of X and y not equal')
            return

        init_w = np.array([-1.43] + [0.01 for i in range(len(X[0])-1)])

        self.w = init_w
        w_acc = np.array([1e-5 for i in range(len(X[0]))])

        # run several passes
        for i in range(self.iteration+1):
            # compute gradients of w and sum over all training data
            wgrad = self.compute_grad(X, y)

            # compute summation of past gradients, for adagrad
            w_acc = w_acc + wgrad**2

            # update parameters, using adagrad
            if self.adagrad:
                self.w = self.w - self.rate*(1.0/np.sqrt(w_acc))*wgrad
            else:
                self.w = self.w - self.rate*wgrad

            # compute and print training error/validation error every 1000 pass
            if i % 1000 == 0:
                # E_in(training set error)
                train_ans = (self.predict(X)>0.5).astype(float)
                train_error = np.mean(np.absolute(y[:,0] - train_ans))
                print('iteration %d,\ttrain error: %f' % (i, train_error))
```

```python
    """
    Predict output values for given data
    params:
        x:  features to predict

    returns:
        y:  output value
    """
    def predict(self, x):
        return expit(np.dot(x, self.w))

    """
    Compute gradients given data
    params:
        X:  training features
        y:  ground truth

    returns:
        wgrad:  gradients for weights
    """
    def compute_grad(self, X, y):
        delta = y[:,0] - self.predict(X)
        wgrad = -delta.dot(X)
        return wgrad
```

(1) predict

$$y = sigmoid\left(\sum\left(w_i \cdot x_i\right)\right)$$

(2) compute grad

計算 w 的 gradient， $grad_i = \sum\left(-x_{ji} \cdot \left(y_j - \bar{y}_j\right)\right)$

(3) update

使用 gradient descent(沒有使用 SGD，一次計算全部 sample)，並用 Adagrad 加強 gradient descent 的功能

2. method 2: Multi-layer Perceptrons

```python
    def fit(self, X, y):
        self.initialize_coef(X, y)

        num_examples = len(X)

        if len(y.shape) == 1:
            y = np.array([[num] for num in y])

        # when should we stop?
        last_error = 1000000.0
        consecutive_increase = 0

        for num_iter in range(self.max_passes):

            a = self.forward(X)

            error = loss(a[-1], y)
            values = np.sum(np.array([np.dot(s.ravel(), s.ravel()) for s in self.W]))
            error_reg = error + (0.5 * self.reg) * values / num_examples

            if num_iter % 50 == 0:
                train_error = np.mean(np.absolute((a[-1]>0.5).astype(float)-y))
                #print('\t', num_iter, '\t', error_reg, '\t', train_error)

                if error_reg > last_error:
                    consecutive_increase += 1
                elif error_reg < last_error:
                    consecutive_increase = 0

                last_error = error_reg

                if consecutive_increase == 2:
                    #print('consecutive increasing, stop training')
                    break

            deltas = [np.array([]) for i in range(len(self.dims))]
            dW, db = [0 for i in range(len(self.W))], [0 for i in range(len(self.b))]

            last = len(self.dims)-2
```

```python
            deltas = [np.array([]) for i in range(len(self.dims))]
            dW, db = [0 for i in range(len(self.W))], [0 for i in range(len(self.b))]

            last = len(self.dims)-2
            deltas[last] = a[-1] - y

            for i in range(len(self.dims)-2, 0, -1):
                deltas[i-1] = np.dot(deltas[i], self.W[i].T)
                derivatives(a[i], deltas[i-1], self.activation_type[i-1])
                dW[i-1] = (np.dot(a[i-1].T, deltas[i-1]) + self.reg * self.W[i-1]) / num_examples
                db[i-1] = np.mean(deltas[i-1], 0)

            grads = dW + db
            self.optimizer.update_params(grads)


    def forward(self, X):
        z, a = [0 for i in range(len(self.dims))], [X]+[0 for i in range(len(self.dims)-1)]

        for i in range(len(self.hidden_layer_size)+1):
            z[i+1] = a[i].dot(self.W[i]) + self.b[i]
            if (i + 1) != len(self.hidden_layer_size)+1:
                a[i+1] = activation(z[i+1], self.activation_type[i])
            else:
                a[i+1] = z[i+1]

        return a
```

第二個方法我選擇實作 DNN，主要的技巧如下：

(1) Forward pass

從 input layer 開始，計算下一層 neuron 的 input $input=activation(b+output\cdot w)$，直到 output layer

(2) Back propagation

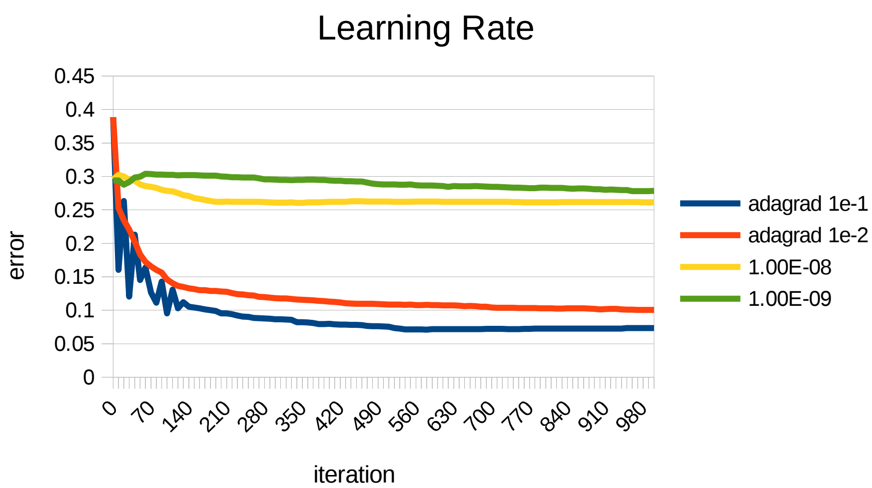
從 output layer 開始，先計算 $delta_{last}=y-\bar{y}$，接著利用每一層間的 w 計算前一層的結果在 delta 上佔的權重，再利用這個權重更新 w

(3) update

使用 ADAM optimizer，參考 sklearn 的實作

3. Discussion

(1) Logistic Regression Learning Rate



沒有使用 adagrad 時，很明顯 w 更新地很慢，甚至會卡住無法下降，使用 adagrad 可以很好地解決這個問題。learning rate 愈大時 error 下降愈快，但 rate 太大時可能會有波動(如 adagrad 1e-1)。

(2) Neural Net Hidden-layer Size

size	(30,1)	(30,30,1)	(30,30,2)	(30,30,5)	(30,30,10)	(30,20,5)	(50,30,5)
accuracy	0.928	0.938	0.943	0.945	0.945	0.946	0.945

針對 DNN 的 hidden layer size 做了一些實驗，這裡的 accuracy 是拿 training data 做 5-fold cross validation 的平均 accuracy。可以看到三層 hidden layer 表現明顯較兩層好，而每層 hidden layer 的維度則沒有太大影響。考慮 training 速度之後，選用(30, 30, 2)來做其他實驗。

(3) Feature Selection

由於在 hw1 中，feature selection 是很重要的一個環節，因此在這次作業也嘗試選擇 feature。

(a) Chi-Square

對於每個 feature 和 class 做 chi-square test，移除值最小的兩維(剛好是最後兩維，平均大寫序列長度和總大寫序列長度)，重新 train 後發現 accuracy 變低。

(b) Pearson correlation

對於每個 feature 和 class 算 pearson correlation，若 feature 和 class 具有線性關係，correlation 應該較大，因此移除 correlation 最小的數維，但實驗後也發現 accuracy 降低。

經過以上兩個實驗後，找不到比較好取 feature 的方法，因此都使用所有維度進行實驗。

(4) Comparison between different models

model	Logistic Regression	MLP	Random Forest n_trees=9	Random Forest n_trees=99	Gradient Boosting n_trees=9	Gradient Boosting n_trees=99
accuracy	0.928	0.945	0.943	0.952	0.928	0.950

除了 DNN，這次還實作了 Random Forest，另外用 sklearn 的 GradientBoostingClassifier 一起做了實驗。除了 Logistic Regression 之外的幾個 model 都有差不多的 accuracy，最後最好的 model 就是用 DNN model + Random Forest model 進行投票得來的。