ML2016_HW2_report

姓名: 徐有慶 學號: r05922162

1. Logistic regression function

Step1. 將 spam_train.csv 中的資料使用矩陣來儲存，形成矩陣 $X \in R^{4001*58}$，其中，spam_train.csv 所提供的 57 個 features 全部用上，並加上一行(column)全部為 1 的行向量給 bias 使用，所以每一列共有 58 維。並將每筆資料的 label 存到矩陣 $Y\_head \in R^{1*4001}$。

Step2. 初始一個矩陣 $W \in R^{1*58}$，其中，每個元素的範圍為 0~0.01，當作初始的權重。

Step3. W*X^T = $Z \in R^{1*4001}$，並將 Z 丟到 sigmoid function 求得 $Y \in R^{1*4001}$

Step4. (Y-Y_head)*X = gradients $\in R^{1*58}$，其中的元素即為每個權重的 gradient。

Step5. Gradient descent

Step6. Y 中元素大於 0.5 則設為 1，反之設為 0，則 $error = \sum_{i=0}^{4001} (y_i - y_{head_i})^2$，其中 $y_i \in$ column i of Y，$y\_head_i \in$ column i of Y_head。

Step7. 重複 Step3~6 直到 error <= 280

Step8. 使用 spam_test.csv 預測資料

Primary code:

```
def training(self):
    weights = self.weights
    x = self.train_set
    z = weights.dot(x.getT())
    y_head = self.y_head
    y = self._sigmoid(z)
    gradients = (y - y_head).dot(x)
```

```
            self.past_gradients.append(gradients)
            self._gradientDescent(gradients)
        return self._error(y, y_head)

    def _gradientDescent(self, gradients):
        learn_rate = 0.1
        sigma_past = np.matrix(np.zeros(self.features_dim))
        for past in self.past_gradients:
            sigma_past = sigma_past + np.power(past,2)
        for i in range(self.weights.size):
            self.weights[0,i] =  self.weights[0,i] - learn_rate * gradients[0,i] /
math.sqrt(sigma_past[0,i])
```

2. Describe your another method, and which one is best

另一方法使用 Probabilistic generative model，機率的分布使用 Gaussian Distribution。

Step1. 將 spam_train.csv 中的資料使用矩陣來儲存，並依照每筆資料的 label 分別存入 class1 及 class0 當中，若 label 為 1 則存入 class1，反之存入 class0，其中，spam_train.csv 所提供的 57 個 features 全部用上，所以 class1 及 class0 中每一列共有 57 維。

Step2. 直接計算出μ1、μ0、Σ

Step3. 使用 spam_test.csv 預測資料

Primary code:

```
def _computeMu(self, class_name):

    mu = np.matrix(np.zeros(self.features_dim))

    for x in class_name:

        x = np.matrix(x)

        mu += x

    mu = mu / len(class_name)

    return mu.getT()



def _computeSigma(self, class_name, mu):

    sigma = np.matrix(np.full((self.features_dim, self.features_dim), 0.0))

    for x in class_name:

        x = np.matrix(x).getT()

        sigma = sigma + ((x-mu).dot((x-mu).getT()))

    sigma = sigma / len(class_name)

    return sigma



def train(self):

    self.u1 = self._computeMu(self.class1)

    self.u0 = self._computeMu(self.class0)

    sigma1 = self._computeSigma(self.class1, self.u1)
```

```
sigma0 = self._computeSigma(self.class0, self.u0)
total = self.N0 + self.N1
self.sigma = (self.N1/total) * sigma1 + (self.N0/total) * sigma0
```

3. Discussion

Logistic regression 的部分利用對 training set 的正確率來停止，並在 gradient descent 的部份使用 adagrad 來加速收斂的速度，初始 learning rate 設為 0.1。Probabilistic generative model 的部分，若 Σ 為 singular matrix 則算其 pseudo inverse 來處理例外狀況。Logistic regression 在 leaderboard public set 上的分數大約在 0.92~0.933 之間，使用 Probabilistic generative model 的話則在 0.87 左右，明顯地，Logistic regression 在此資料上表現較佳。下表為不同 training set 的大小對 Probabilistic generative model 在 test set 上結果的影響。

Training set size	Private set score
500 筆	0.85333
1000 筆	0.86333
2000 筆	0.85667
3000 筆	0.86000
4000 筆	0.86000

結果顯示在此資料上使用 Probabilistic generative model 做訓練，training set 的大小對結果影響不大。