Recitation #6
CIS 519
CIS 519 TA Team
Algorithm 1 Perceptron

1: Initial weight vector: \( w_1 = 0 \in \mathbb{R}^d \)
2: for \( t = 1 \rightarrow T \) do
3: \hspace{1em} Receive instance \( x_t \in X \subseteq \mathbb{R}^d \)
4: \hspace{1em} Predict \( \hat{y} = \text{sign}(w_t^T x_t) \)
5: \hspace{1em} Receive true label \( y_t \in \{\pm 1\} \)
6: \hspace{1em} Incur loss \( 1(\hat{y}_t \neq y_t) \)
7: \hspace{1em} Update:
8: \hspace{2em} if \( \hat{y}_t \neq y_t \) then
9: \hspace{3em} \( w_{t+1} \leftarrow w_t + y_t x_t \)
10: \hspace{2em} else
11: \hspace{3em} \( w_{t+1} \leftarrow w_t \)
12: \hspace{1em} end if
13: end for
Algorithm 2 Winnow

1: Learning rate parameter $\eta > 0$
2: Initial weight vector: $w_1 = (\frac{1}{d}, ..., \frac{1}{d}) \in \mathbb{R}^d$
3: for $t = 1 \rightarrow T$ do
4:    Receive instance $x_t \in X \subseteq \mathbb{R}^d$
5:    Predict $\hat{y} = \text{sign}(w_t^T x_t)$
6:    Receive true label $y_t \in \{\pm 1\}$
7:    Incur loss $1(\hat{y}_t \neq y_t)$
8:    Update:
9:    if $\hat{y}_t \neq y_t$ then
10:        for $i = 1 \rightarrow d$ do
11:            $w_{t+1,i} \leftarrow \frac{w_{t,i} \exp(\eta y_t x_{t,i})}{Z_t}$
12:            where $Z_t = \sum_{i=1}^{d} w_{t,i} \exp(\eta y_t x_{t,i})$
13:        end for
14:    else
15:        $w_{t+1} \leftarrow w_t$
16:    end if
17: end for
Perceptron with AdaGrad

\[ g_t = \begin{cases} 
0 & \text{if } y(w_t^T x + \theta) > 1 \\
-y(x, 1) & \text{otherwise}
\end{cases} \]

That is, for the first n features, that gradient is \(-yx\), and for \(\theta\), it is always \(-y\).

Then, for each feature \(j\) (\(j = 1, ..., n + 1\)) we keep the sum of the gradients’ squares:

\[ G_{t,j} = \sum_{k=1}^{t} g_{k,j}^2 \]

and the update rule is

\[ w_{t+1,j} \leftarrow w_{t,j} - \eta g_{t,j} / (G_{t,j})^{1/2} \]

By substituting \(g_t\) into the update rule above, we get the final update rule:

\[ w_{t+1,j} = \begin{cases} 
w_{t,j} & \text{if } y(w_t^T x + \theta) > 1 \\
w_{t,j} + \eta y x_j / (G_{t,j})^{1/2} & \text{otherwise}
\end{cases} \]
Averaged Perceptron

Algorithm Averaged Perceptron

1: Training:
2: [m: #(examples); k: #(mistakes) = #(hypotheses); c_i: consistency count for v_i ]
3: Input: a labeled training set \((x_1, y_1), \ldots, (x_m, y_m)\), Number of epochs T
4: Output: a list of weighted perceptrons \((v_1, c_1), \ldots, (v_k, c_k)\)
5: Initialize: k=0; v_1 = 0, c_1 = 0
6: Repeat T times:
7: for \( t = 1 \rightarrow m \) do
8: Compute prediction \( \hat{y} = \text{sgn}(v_k \cdot x_i) \)
9: if \( \hat{y} = y_i \) then
10: \( c_k = c_k + 1 \)
11: else
12: \( v_{k+1} = v_k + y_i x \)
13: \( c_{k+1} = 1 \)
14: \( k = k + 1 \)
15: end if
16: end for
17: Prediction:
18: Given: a list of weighted perceptrons \((v_1, c_1), \ldots, (v_k, c_k)\), a new example x
19: Predict: the label(x) as follows:
20: \( y(x) = \text{sgn}[\sum_{i=1}^{k} c_i v_i \cdot x] \)
Averaged Perceptron Implementation Details

• This average should be implemented by keeping only two weight vector. A cumulative weight vector computed during the training, and the current one.
Understand the code

- Readers
- Perceptron Classifier
- Feature Extraction
Real-world Reader

# Parse the real-world data to generate features,
# Returns a list of tuple lists

def parse_real_data(path):
    # List of tuples for each sentence
    data = []
    for filename in os.listdir(path):
        with open(path+filename, 'r') as file:
            sentence = []
            for line in file:
                if line == '\n':
                    data.append(sentence)
                    sentence = []
                else:
                    sentence.append(tuple(line.split('')))
    return data
def parse_synthetic_labels(path):
    # List of tuples for each sentence
    labels = []
    with open(path+'y.txt', 'rb') as file:
        for line in file:
            labels.append(int(line.strip()))
    return labels
def parse.synthetic.data(path):
    # List of tuples for each sentence
    data = []
    with open(path+'x.txt') as file:
        features = []
        for line in file:
            #print('Line:', line)
            for ch in line:
                if ch == '[' or ch.isspace():
                    continue
                elif ch == ']':
                    data.append(features)
                    features = []
                else:
                    features.append(int(ch))
    return data
Test Real-world Reader

- email_dev_data = parse_real_data('Data/Real-World/Enron/dev/')

- news_dev_data = parse_real_data('Data/Real-World/CoNLL/dev/')
Test Synthetic Reader

- `syn_dense_dev_data = parse_synthetic_data('Data/Synthetic/Dense/dev/')`

- `syn_dense_dev_labels = parse_synthetic_labels('Data/Synthetic/Dense/dev/')`

- `syn_sparse_dev_data = parse_synthetic_data('Data/Synthetic/Sparse/dev/')`

- `syn_sparse_dev_labels = parse_synthetic_labels('Data/Synthetic/Sparse/dev/')`
class Classifier(object):
    
    def __init__(self, algorithm, x_train, y_train, iterations=1, averaged = False, eta = 1, alpha = 1):
        
        # Get features from examples; this line figures out what features are present in
        # the training data, such as 'w-1=dog' or 'w+1=cat'
        features = {feature for xi in x_train for feature in xi.keys()}

        if algorithm == 'Perceptron':
            # Initialize w, bias
            self.w, self.w['bias'] = {feature:0.0 for feature in features}, 0.0
            # Iterate over the training data n times
            for i in range(iterations):
                # Check each training example
                for i in range(len(x_train)):
                    xi, yi = x_train[i], y_train[i]
                    y_hat = self.predict(xi)
                    # Update weights if there is a misclassification
                    if yi != y_hat:
                        for feature, value in xi.items():
                            self.w[feature] = self.w[feature] + yi*eta*value
                            self.w['bias'] = self.w['bias'] + yi*eta

    def predict(self, x):
        s = sum([self.w[feature]*value for feature, value in x.items()]) + self.w['bias']
        return 1 if s > 0 else -1
# Feature extraction
print('Extracting features from real-world data...')
news_train_y = []
news_train_x = []
train_features = set([])
for sentence in news_train_data:
    padded = sentence[:]
    padded.insert(0, ('pad', None))
    padded.append(('pad', None))
    for i in range(1, len(padded)-1):
        news_train_y.append(1 if padded[i][1]=='I' else -1)
        feat1 = 'w-1='+str(padded[i-1][0])
        feat2 = 'w+1='+str(padded[i+1][0])
        feats = [feat1, feat2]
        train_features.update(feats)
        feats = {feature:1 for feature in feats}
    news_train_x.append(feats)
Thanks!