Recitation #6
CIS 519

CIS 519 TA Team

Perceptron

Algorithm 1 Perceptron

Initial weight vector: $\mathbf{w_1} = \mathbf{0} \in R^d$
for $t = 1 \rightarrow T$ **do**
    Receive instance $\mathbf{x_t} \in \mathbf{X} \subseteq R^d$
    Predict $\hat{y} = sign(\mathbf{w_t}^T \mathbf{x_t})$
    Receive true label $y_t \in \{\pm 1\}$
    Incur loss $\mathbf{1}(\hat{y}_t \neq y_t)$
    Update:
    if $\hat{y}_t \neq y_t$ **then**
        $\mathbf{w_{t+1}} \leftarrow \mathbf{w_t} + y_t \mathbf{x_t}$
    else
        $\mathbf{w_{t+1}} \leftarrow \mathbf{w_t}$
    end if
end for

Winnow

Algorithm 2 Winnow

Learning rate parameter $\eta > 0$
Initial weight vector: $\mathbf{w_1} = (\frac{1}{d}, ..., \frac{1}{d}) \in R^d$
for $t = 1 \rightarrow T$ **do**
    Receive instance $\mathbf{x_t} \in \mathbf{X} \subseteq R^d$
    Predict $\hat{y} = sign(\mathbf{w_t}^T \mathbf{x_t})$
    Receive true label $y_t \in \{\pm 1\}$
    Incur loss $\mathbf{1}(\hat{y}_t \neq y_t)$
    Update:
    if $\hat{y}_t \neq y_t$ **then**
        for $i = 1 \rightarrow d$ **do**
            $w_{t+1,i} \leftarrow \frac{w_{t,i} exp(\eta y_t x_{t,i})}{Z_t}$
            where $Z_t = \sum_{j=1}^{n} w_{t,j} exp(\eta y_t x_{t,j})$
        end for
    else
        $\mathbf{w_{t+1}} \leftarrow \mathbf{w_t}$
    end if
end for

Perceptron with AdaGrad

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

That is, for the first n features, that gradient is $-yx$, and for θ, 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:

$$w_{t+1,j} = \begin{cases} w_{t,j} & \text{if } y(w_t^\intercal x + \theta) > 1 \\ w_{t,j} + \eta y x_j / (G_{t,j})^{\frac{1}{2}} & otherwise \end{cases}$$

Averaged Perceptron

Algorithm Averaged Perceptron

Training:
[m: #(examples); k: #(mistakes) = #(hypotheses); c_i: consistency count for v_i]
Input: a labeled training set (x_1, y_1),...(x_m, y_m), Number of epochs T
Output: a list of weighted perceptrons $(v_1, c_1), ..., (v_k, c_k)$
Initialize: k=0; $v_1 = 0$, $c_1 = 0$
Repeat T times:
for $t = 1 \rightarrow m$ **do**
    Compute prediction $\hat{y} = sgn(v_k \cdot x_i)$
    if $\hat{y} = y_i$ **then**
        $c_k = c_k + 1$
    else
        $v_{k+1} = v_k + y_i x$
        $c_{k+1} = 1$
        $k = k + 1$
    end if
end for
Prediction:
Given: a list of weighted perceptrons $(v_1, c_1), ...(v_k, c_k)$, a new example x
Predict: the label(x) as follows:
$y(x) = sgn[\sum_1^k c_i \, v_i \cdot x]$

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
```

Synthetic Reader 1

```
#Returns a list of labels
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
```

Synthetic Reader 2

```
#Returns a list of features
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/')

Perceptron Classifier

```
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

```
# 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!