Introduction of SVM^{struct}

Hung-yi Lee

SVM^{struct}

- API for implementing different kinds of structured learning tasks by structured SVM
  - C: https://www.cs.cornell.edu/people/tj/svm_light/svm_struct.html
  - Python: http://tfinley.net/software/svmpython2/
  - MATLAB: http://www.robots.ox.ac.uk/~vedaldi/svmstruct.html
- Structured SVM with Hidden Information
  - http://www.cs.cornell.edu/~cnyu/latentssvm/

SVM^{struct}

- Python version in these slides
- Download from http://tfinley.net/software/svmpython2/svm-python-v204.tgz
- To build this, a simple make should work.
  - *svm_python_learn*: for learning a model and
  - *svm_python_classify*: for testing with a learned model

svm_python_learn

What you have to implement by yourself

$\{(x^1, \hat{y}^1), (x^2, \hat{y}^2), \cdots, (x^N, \hat{y}^N)\} =$ read_examples(TrainingData)

init_model()

w ← 0, working set $\mathbb{A}^1 \leftarrow null, \mathbb{A}^2 \leftarrow null, \cdots, \mathbb{A}^N \leftarrow null$

Repeat

For each training data**:**

$\bar{y}^n =$ find_most_violated_constraint($x^n, \hat{y}^n$)

$$\arg\max_y [\Delta(\hat{y}^n, y) + w \cdot \phi(x^n, y)]$$

Update working set $\mathbb{A}^n \leftarrow \mathbb{A}^n \cup \{\bar{y}^n\}$

$w \leftarrow$Solve a **QP** with Working Set $\mathbb{A}^1, \mathbb{A}^2, \cdots, \mathbb{A}^N$

Until the stopping criterion fulfilled

Return w

svm_python_learn - QP

QP: Find w, $\varepsilon^1 \ldots \varepsilon^N$ minimizing $\frac{1}{2}\|w\|^2 + \lambda \sum_{n=1}^{N} \varepsilon^n$

For $\forall n$:

For $\forall y \in \mathbb{A}^n$:

$$w \cdot \left(\phi(x^n, \hat{y}^n) - \phi(x^n, y)\right) \geq \Delta(\hat{y}^n, y) - \varepsilon^n, \varepsilon^n \geq 0$$

psi(x, y)

loss$(\hat{y}^n, y)$

Usage: ./svm_python_learn –c 1000 TrainingData Model

-e value (smaller, training time longer)

-k int (number of new constraints to accumulate before recomputing the QP)

svm_python_classify

For each test data**:**

$\tilde{y}' \leftarrow$ classify_example (x^n) Inference

$$\arg\max_{y} w \cdot \phi(x^n, y)$$

Usage: ./svm_python_classify TestingData Model Result

svmstruct.py

- Put your code in ***svmstruct.py***

read_examples(filename)

init_model()

find_most_violated_constraint($x^n, \hat{y}^n$)

psi(x, y)

loss($\hat{y}^n, y$)

classify_example (x^n)

Using multi-class classification to show how to implement each function

http://tfinley.net/software/svmpython2/multiclass_code.html

read_examples

$\{(x^1, \hat{y}^1), (x^2, \hat{y}^2), \cdots, (x^N, \hat{y}^N)\} =$ read_examples(TrainingData)

```
def read_examples(filename, sparm):
    """Parses an input file into an example sequence."""
    # This reads example files of the type read by SVM^multiclass.
    examples = []
    # Open the file and read each example.
    for line in file(filename):
        # Get rid of comments.
        if line.find('#'): line = line[:line.find('#')]
        tokens = line.split()
        # If the line is empty, who cares?
        if not tokens: continue
        # Get the target.
        target = int(tokens[0])
        # Get the features.
        tokens = [tuple(t.split(':')) for t in tokens[1:]]
        features = [(0,1)]+[(int(k),float(v)) for k,v in tokens]
        # Add the example to the list
        examples.append((features, target))
    # Print out some very useful statistics.
    print len(examples),'examples read'
    return examples
```

read_examples

- Format of “examples” (training data)
  - The whole “examples” (training data) is a list
  - Each data is a tuple with two element

$$examples = [\ \ (x^1, \hat{y}^1), \ldots\ \ (x^n, \hat{y}^n), \ldots\ \ (x^N, \hat{y}^N)\ \ ]$$

Input object

output object

Sparse representation of a vector

```
[
(3,6.55),
(7,-1.01),
…
(10,0.4)
]
```

a scalar

init_model

- Tell SVM^{struct} the feature dimension of w

Model

```
def init_model(sample, sm, sparm):
    sm.num_features = max(max(x) for x,y in sample)[0]+1
    print 'num_features set to',sm.num_features
    sm.num_classes = max(y for x,y in sample)
    print 'num_classes set to', sm.num_classes
    sm.size_psi = sm.num_features * sm.num_classes
    print                                  m.size_psi
```

feature dimension of w

e.g. K classes, feature representation of x is M dimensions

sm.size_psi = K * M

loss

- $\Delta(\hat{y}, y)$

```
def loss(y, ybar, sparm):
    if y != ybar:
        return 1.0
    else:
        return 0.0
```

Task dependent

psi

$$\phi(x, y) = \begin{bmatrix} 0 \\ \vdots \\ \vec{x} \\ \vdots \\ 0 \end{bmatrix}$$

$K \times M$ dims

K blocks

Each has M dimensions

If x is class k, put $\vec{x}$ at kth block

0 for other dimensions

y = k (class k)

x:

[
(3,6.55),
(7,-1.01),
…
(10,0.4)
]

[
(3+**(k-1)*M**,6.55),
(7+**(k-1)*M**,-1.01),
…
(10+**(k-1)*M**,0.4)
]

vec = svmapi.Sparse( )

return vec

```
def psi(x, y, sm, sparm):
    """Returns the combined feature vector Psi(x,y)."""
    # Just increment the feature index to the appropriate stack position.
    offset = sm.num_features * (y-1)
    pvec = svmapi.Sparse([(k+offset,v) for k,v in x])
    return pvec
```

find_most_violated_constraint

$$\bar{y}^n = \arg\max_y [\Delta(\hat{y}^n, y) + w \cdot \phi(x^n, y)]$$

```
def find_most_violated_constraint(x, y, sm, sparm):
    scores = [(classification_score(x,c,sm,sparm)+loss(y,c,sparm), c)
              for c in xrange(1,sm.num_classes+1)]
    return max(scores)[1]
```

```
def classification_score(x,y,sm,sparm):
    return sum([ sm.w[k]*v for k,v in psi(x,y,sm,sparm) ])
```

vec = psi(x,y)

w = sm.w

score = 0 $w \cdot \phi(x^n, y)$

for k, v in vec: The value at k dimension is v.

score = score + v * w[k]

classify_example

For each test data**:**

$\tilde{y}' \leftarrow$ classify_example (x^n) Inference

$$\arg\max_y w \cdot \phi(x^n, y)$$

```
def classify_example(x, sm, sparm):
    scores = [(classification_score(x,c,sm,sparm), c)
              for c in xrange(1,sm.num_classes+1)]
    return max(scores)[1]
```

$$\bar{y}^n = \arg\max_y [\Delta(\hat{y}^n, y) + w \cdot \phi(x^n, y)]$$