COMP4680: Advanced Topics in Statistical Machine Learning
Third Programming Assignment

Manab Chetia (u5492350), Libo Yin (u5483742)
The Australian National University

September 9, 2015

1 Conditional Random Field Modeling

(a) First consider a single data point $\langle D, \vec{y} \rangle \in D$. By definition:

$$P(\vec{y}|X) = \frac{1}{Z(X)} \exp\left\{ \sum_{j=1}^{m} \langle w_{\vec{y}_j}, X_j \rangle + \sum_{j=1}^{m-1} T_{\vec{y}_j, \vec{y}_{j+1}} \right\}$$

where

$$Z(X) = \sum_{\hat{y} \in y^m} \exp\left\{ \sum_{j=1}^{m} \langle w_{\hat{y}_j}, X_j \rangle + \sum_{j=1}^{m-1} T_{\hat{y}_j, \hat{y}_{j+1}} \right\}$$

So the logarithm is:

$$\log P(\vec{y}|X) = \sum_{j=1}^{m} \langle w_{\vec{y}_j}, X_j \rangle + \sum_{j=1}^{m-1} T_{\vec{y}_j, \vec{y}_{j+1}} - \log \sum_{\hat{y} \in y^m} \exp\left\{ \sum_{j=1}^{m} \langle w_{\hat{y}_j}, X_j \rangle + \sum_{j=1}^{m-1} T_{\hat{y}_j, \hat{y}_{j+1}} \right\}$$

So the gradient is:

$$\begin{aligned} \nabla_{w_Y} \log P(\vec{y}|X) &= \nabla_{w_Y} \sum_{j=1}^{m} \langle w_{\vec{y}_j}, X_j \rangle - \frac{1}{Z(X)} \nabla_{w_Y} \sum_{\hat{y} \in y^m} \exp\left\{ \sum_{j=1}^{m} \langle w_{\hat{y}_j}, X_j \rangle + \sum_{j=1}^{m-1} T_{\hat{y}_j, \hat{y}_{j+1}} \right\} \\ &= \sum_{j=1}^{m} I[Y = \vec{y}_j] \vec{x}_j - \frac{1}{Z(X)} \sum_{\hat{y} \in y^m} \nabla_{w_Y} \exp\left\{ \sum_{j=1}^{m} \langle w_{\hat{y}_j}, X_j \rangle + \sum_{j=1}^{m-1} T_{\hat{y}_j, \hat{y}_{j+1}} \right\} \end{aligned}$$

where Y is a random variable, and

$$\begin{aligned} \nabla_{w_Y} \exp\left\{ \sum_{j=1}^{m} \langle w_{\hat{y}_j}, X_j \rangle + \sum_{j=1}^{m-1} T_{\hat{y}_j, \hat{y}_{j+1}} \right\} &= \exp\left\{ \sum_{j=1}^{m} \langle w_{\hat{y}_j}, X_j \rangle + \sum_{j=1}^{m-1} T_{\hat{y}_j, \hat{y}_{j+1}} \right\} \nabla_{w_Y} \sum_{j=1}^{m} \langle w_{\hat{y}_j}, X_j \rangle \\ &= \exp\left\{ \sum_{j=1}^{m} \langle w_{\hat{y}_j}, X_j \rangle + \sum_{j=1}^{m-1} T_{\hat{y}_j, \hat{y}_{j+1}} \right\} \sum_{j=1}^{m} I[Y = \hat{y}_j] X_j \end{aligned}$$

Therefore:

$$
\begin{aligned}
&\nabla_{w_Y} \log P(\vec{y}\,|X) \\
&= \sum_{j=1}^{m} I[Y = \vec{y}_j] X_j - \frac{1}{Z(X)} \sum_{\hat{y} \in y^m} \exp\left\{ \sum_{j=1}^{m} \langle w_{\hat{y}_j}, X_j \rangle + \sum_{j=1}^{m-1} T_{\hat{y}_j, \hat{y}_{j+1}} \right\} \sum_{j=1}^{m} I[Y = \hat{y}_j] X_j \\
&= \sum_{j=1}^{m} I[Y = \vec{y}_j] X_j - \sum_{\hat{y} \in y^m} P(\hat{y}|X) \sum_{j=1}^{m} I[Y = \hat{y}_j] X_j \\
&= \sum_{j=1}^{m} I[Y = \vec{y}_j] X_j - E_{\hat{y}|X} \left[\sum_{j=1}^{m} I[Y = \hat{y}_j] X_j \right]
\end{aligned}
$$

Similarly:

$$
\begin{aligned}
&\nabla_T \log P(\vec{y}\,|X) \\
&= \nabla_T \sum_{j=1}^{m-1} T_{\vec{y}_j, \vec{y}_{j+1}} - \nabla_T \log \sum_{\hat{y} \in y^m} \exp\left\{ \sum_{j=1}^{m} \langle w_{\hat{y}_j}, X_j \rangle + \sum_{j=1}^{m-1} T_{\hat{y}_j, \hat{y}_{j+1}} \right\} \\
&= [I(\langle i, j \rangle \in \vec{y})]_{\langle i,j \rangle \in 26 \times 26} - \frac{1}{Z(X)} \sum_{\hat{y} \in y^m} \nabla_T \exp\left\{ \sum_{j=1}^{m} \langle w_{\hat{y}_j}, X_j \rangle + \sum_{j=1}^{m-1} T_{\hat{y}_j, \hat{y}_{j+1}} \right\} \\
&= [I(\langle i, j \rangle \in \vec{y})]_{\langle i,j \rangle \in 26 \times 26} - \frac{1}{Z(X)} \sum_{\hat{y} \in y^m} \exp\left\{ \sum_{j=1}^{m} \langle w_{\hat{y}_j}, X_j \rangle + \sum_{j=1}^{m-1} T_{\hat{y}_j, \hat{y}_{j+1}} \right\} \nabla_T \sum_{j=1}^{m-1} T_{\hat{y}_j, \hat{y}_{j+1}} \\
&= [I(\langle i, j \rangle \in \vec{y})]_{\langle i,j \rangle \in 26 \times 26} - \sum_{\hat{y} \in y^m} P(\hat{y}|X) [I(\langle i, j \rangle \in \hat{y})]_{\langle i,j \rangle \in 26 \times 26} \\
&= [I(\langle i, j \rangle \in \vec{y})]_{\langle i,j \rangle \in 26 \times 26} - E_{\hat{y}|X} [I(\langle i, j \rangle \in \hat{y})]_{\langle i,j \rangle \in 26 \times 26}
\end{aligned}
$$

where $[I(\langle i, j \rangle \in \vec{y})]_{\langle i,j \rangle \in 26 \times 26}$ denotes a 26×26 matrix whose value on coordinate $\langle i, j \rangle$ is 1 if $\vec{y}$ contains transition $\langle i, j \rangle$,and 0 otherwise. The above derivations naturally extend to all data points:

$$
\begin{aligned}
&\log \prod_{\langle \vec{x}, \vec{y} \rangle \in D} P(\vec{y}\,|\vec{x}) = \sum_{\langle \vec{x}, \vec{y} \rangle \in D} \log P(\vec{y}\,|\vec{x}) \\
&= \sum_{\langle \vec{x}, \vec{y} \rangle \in D} \left[\sum_{j=1}^{m} \langle w_{\vec{y}_j}, \vec{x}_j \rangle + \sum_{j=1}^{m-1} T_{\vec{y}_j, \vec{y}_{j+1}} - \log \sum_{\hat{y} \in y^m} \exp\left(\sum_{j=1}^{m} \langle w_{\hat{y}_j}, \vec{x}_j \rangle + \sum_{j=1}^{m-1} T_{\hat{y}_j, \hat{y}_{j+1}} \right) \right]
\end{aligned}
$$

$$
\begin{aligned}
\nabla_{w_Y} \log \prod_{\langle \vec{x}, \vec{y} \rangle \in D} P(\vec{y}\,|\vec{x}) &= \nabla_{w_Y} \sum_{\langle \vec{x}, \vec{y} \rangle \in D} \log P(\vec{y}\,|\vec{x}) \\
&= \sum_{\langle \vec{x}, \vec{y} \rangle \in D} \nabla_{w_Y} \log P(\vec{y}\,|\vec{x}) \\
&= \sum_{\langle \vec{x}, \vec{y} \rangle \in D} \sum_{j=1}^{m} I(Y = \vec{y}_j) \vec{x}_j - E_{\hat{y}|\vec{x}} \left[\sum_{j=1}^{m} I(Y = \hat{y}_j) \vec{x}_j \right]
\end{aligned}
$$

$$\begin{aligned}\nabla_T \log \prod_{\langle \vec{x},\vec{y}\rangle \in D} P(\vec{y}\,|\vec{x}) &= \nabla_T \sum_{\langle \vec{x},\vec{y}\rangle \in D} \log P(\vec{y}\,|\vec{x}) \\ &= \sum_{\langle \vec{x},\vec{y}\rangle \in D} \nabla_T \log P(\vec{y}\,|\vec{x}) \\ &= \sum_{\langle \vec{x},\vec{y}\rangle \in D} [I(\langle i,j\rangle \in \vec{y})]_{\langle i,j\rangle \in 26\times 26} - E_{\hat{y}|\vec{x}}[I(\langle i,j\rangle \in \hat{y})]_{\langle i,j\rangle \in 26\times 26}\end{aligned}$$

(b) From the derivations above, we can see that the features are:

$$\sum_{j=1}^{m} I(Y = y_j)\vec{x}_j \text{ and } [I(\langle i,j\rangle \in y)]_{\langle i,j\rangle \in 26\times 26}$$

(c) The decoder uses the max-sum algorithm. The code is located at `CRF/Decoder.py`, and the result is located at `result/decode_output.txt`. The recursive relation of the algorithm is shown as follows:

$$\begin{aligned}\arg\max_{\vec{y}} P(\vec{y}\,|\vec{x}) &= \arg\max_{\vec{y}} \frac{1}{Z(\vec{x})} \exp\left(\sum_{j=1}^{m}\langle w_{\vec{y}_j}, \vec{x}_j\rangle + \sum_{j=1}^{m-1} T_{\vec{y}_j,\vec{y}_{j+1}}\right) \\ &\qquad \text{since } Z(\vec{x}) \text{ is a constant, and the exponential function is monotonic} \\ &= \arg\max_{\vec{y}} \left(\sum_{j=1}^{m}\langle w_{\vec{y}_j}, \vec{x}_j\rangle + \sum_{j=2}^{m} T_{\vec{y}_{j-1},\vec{y}_j}\right) \\ &= \arg\max_{\vec{y}_m} \left(\underbrace{\arg\max_{\vec{y}_{1:m-1}} \left(\sum_{j=1}^{m-1}\langle w_{\vec{y}_j}, \vec{x}_j\rangle + \sum_{j=2}^{m-1} T_{\vec{y}_{j-1},\vec{y}_j}\right)}_{\text{recursion}} + \langle w_{\vec{y}_m}, \vec{x}_m\rangle + T_{\vec{y}_{m-1},\vec{y}_m}\right)\end{aligned}$$

The max operator follows the same recursion, and the maximum value of the objective function $\sum_{j=1}^{m}\langle w_{\vec{y}_j}, \vec{x}_j\rangle + \sum_{j=1}^{m-1} T_{\vec{y}_j,\vec{y}_{j+1}}$ is 200.18515. The max-sum algorithm reduces the complexity from $O(|y|^m)$ to $O(m|y|^2)$ by caching intermediate results. Interestingly, this algorithm does not use the undirected property of the model.

2 Training Conditional Random Fields

(a) To calculate $\log P(\vec{y}|\vec{x})$, we need to calculate $Z(\vec{x})$. Here, we combine the ideas from message passing and dynamic programming. Imagine a sequence of m letters in a word as a left-to-right path through a $|y| \times m$ lattice. At each step, the path extends exactly one column rightwards. Therefore, at each node in this lattice, there are edges from every node on the left column, and edges to every nodes on the right column. Therefore, if we swipe this lattice from left to right, following all possible $|y| \times |y|$ edges at each column transition, we can calculate the sum of all partial, unnormalized probabilities of partial sequences that end on a certain letter. To calculate $Z(\vec{x})$, we simply sum the rightmost

column. The recursive relation of the algorithm is shown as follows:

$$
\begin{aligned}
Z(\vec{x}) &= \sum_{\hat{y}\in y^m} \exp\left(\sum_{j=1}^{m}\langle w_{\hat{y}_j}, \vec{x}_j\rangle + \sum_{j=1}^{m-1} T_{\hat{y}_j,\hat{y}_{j+1}}\right) \\
&= \sum_{\hat{y}\in y^m} \left(\prod_{j=1}^{m} \exp\left(\langle w_{\hat{y}_j}, \vec{x}_j\rangle\right) \prod_{j=2}^{m} \exp(T_{\hat{y}_{j-1},\hat{y}_j})\right) \\
&= \sum_{\hat{y}_m} \left(\exp\left(\langle w_{\hat{y}_j}, \vec{x}_j\rangle\right) \exp(T_{\hat{y}_{j-1},\hat{y}_j}) \sum_{\hat{y}_{1:m-1}\in y^{m-1}} \prod_{j=1}^{m-1} \exp\left(\langle w_{\hat{y}_j}, \vec{x}_j\rangle\right) \prod_{j=2}^{m-1} \exp(T_{\hat{y}_{j-1},\hat{y}_j})\right) \\
&= \sum_{\hat{y}_m} \left(\exp\left(\langle w_{\hat{y}_j}, \vec{x}_j\rangle + T_{\hat{y}_{j-1},\hat{y}_j}\right) \underbrace{\sum_{\hat{y}_{1:m-1}\in y^{m-1}} \exp\left(\sum_{j=1}^{m-1}\langle w_{\hat{y}_j}, \vec{x}_j\rangle + \sum_{j=2}^{m-1} T_{\hat{y}_{j-1},\hat{y}_j}\right)}_{\text{recursion}}\right)
\end{aligned}
$$

By caching intermediate results, $Z(\vec{x})$ can be calculated in $O(m|y|^2)$, same as the max-sum algorithm. Finally, $\frac{1}{n}\sum_{i=1}^{n} \log P(\vec{y}|\vec{x}) = -31.2884$. Strangely, we did not find any place where we could use the log-sum-exp trick during the implementation.

The hard part in calculating $\nabla_w \log P(\vec{y}\,|\vec{x})$ and $\nabla_T \log P(\vec{y}\,|\vec{x})$ is the expectation of some indicator functions on the true distribution $P(\hat{y}|\vec{x})$. For ∇_w, the indicator function detects weather a word contains a certain letter; for ∇_T, the indicator function detects whether a word contains a certain transition. In other words, the task is to efficiently calculate the sum of the probability of all possible words that meet certain criteria. The key observation here is that because the model is undirected, the aforementioned forward algorithm is equivalent to a backward algorithm that swipes the lattice from right to left. By multiplying the sum of partial probabilities of all sequences from both sides that meet on a single node or an edge, we can get the sum of possibilities of all sequences that goes through that node or edge. This "multiplicability" is guaranteed by the distributivity of multiplication on addition. We skip the mathematical representation of recursive relation here since the notation is too complected. Finally, the code is located at `CRF/Gradient.py`, and the result is located at `result/gradient.txt`.

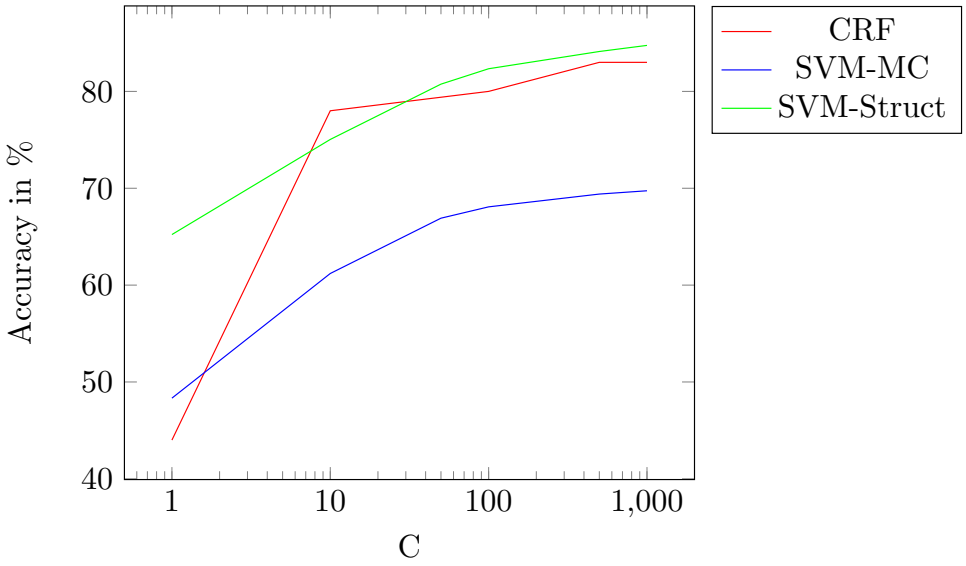
(b) In this task, we use the `fmin_l_bfgs_b` framework from `numpy.optimize`. We left all parameters by default except `func`, `fprime`, and `x0`. With C=1000, the minimum value of the objective function is 3701.158. The training and testing code are located at `CRF/TrainCRF/learn.py` and `CRF/TrainCRF/test.py`, respectively, and the result is located at `result/solution.txt` and `result/prediction.txt`.

Interestingly, although it took less than 1 minute to obtain the gradient for all data points (we did not manage to reach the 5 seconds benchmark, though), the entire training process with C=1000 took almost 3 hours. The training can be made faster with smaller C values.

3 Benchmarking with other methods

3.1 Accuracy on letter wise prediction

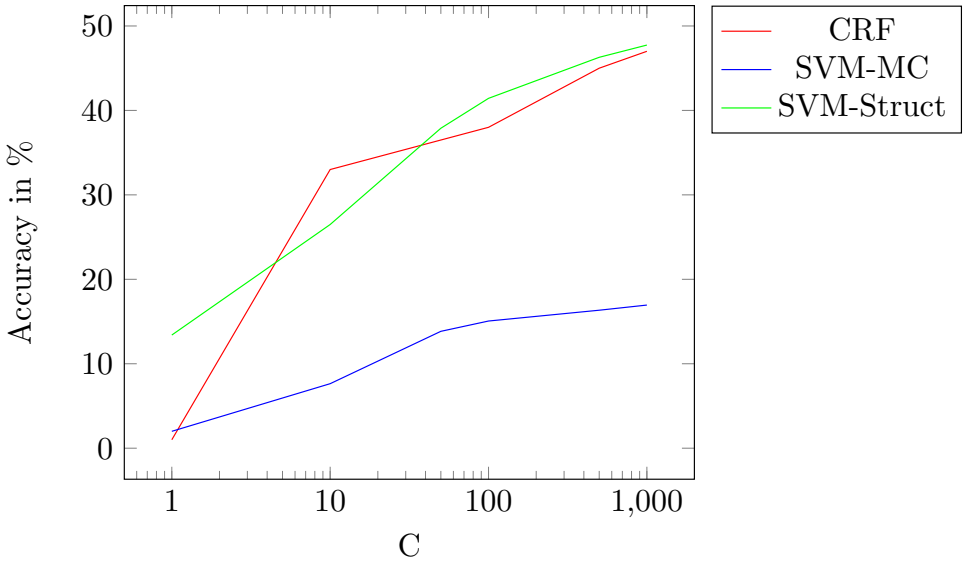
Letter wise accuracy using CRF, SVM-MC and SVM-Struct



Observation: SVM-Struct performs slightly better than CRF, whereas SVM-MC performs poorly. SVM-Struct performs better as it takes into consideration how much we dislike between the predicted and true output and as expected SVM-MC won't perform better as it takes into consideration only individual letters and not it's neighbourhood.

3.2 Accuracy on word wise prediction

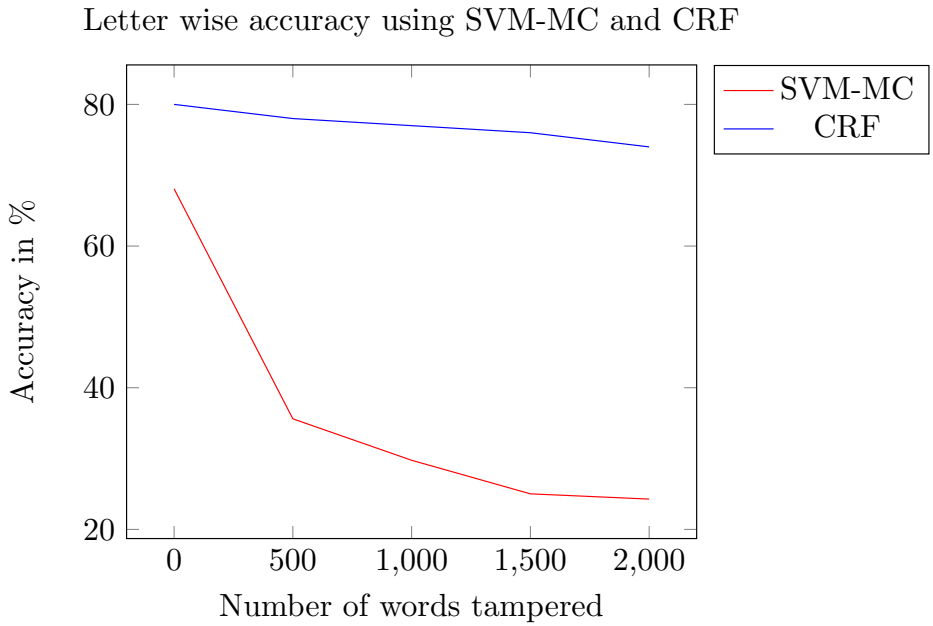
Word wise accuracy using CRF, SVM-MC, and SVM-Struct



Observation: Again SVM-Struct and CRF outperforms SVM-MC and SVM-Struct performs slightly better than CRF due to the same above mentioned reasons and as we increase C the accuracy increases as we are able to fit our data more and more.

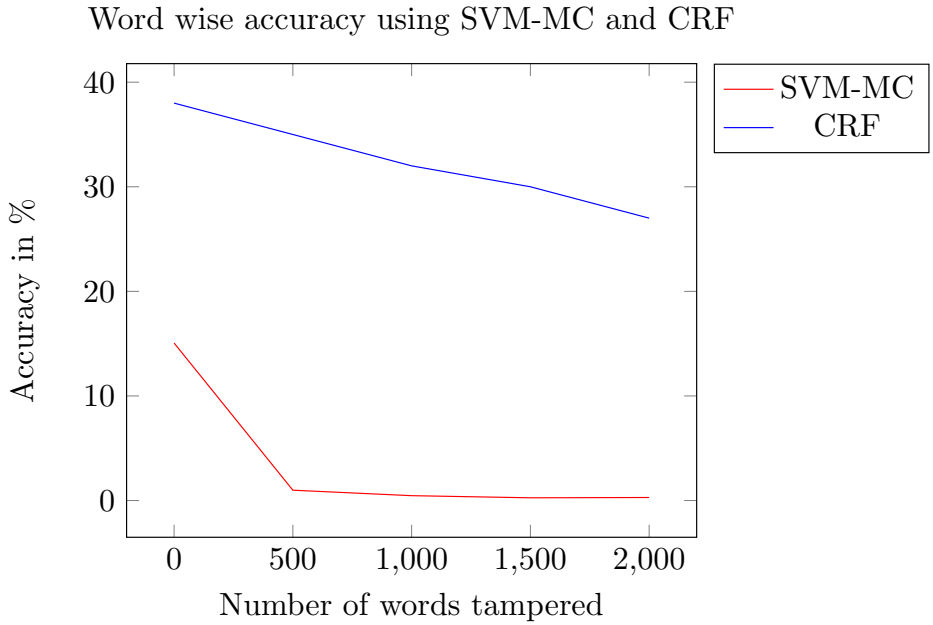
4 Robustness to Tampering

4.1 Accuracy on letter wise prediction



Observation: CRF does much better than SVM-MC even when letters are tampered and the accuracy for CRF is more than 70% which is more than 2 times the accuracy for SVM-MC. This shows the advantages of using CRF over SVM-MC as it takes into context of its neighbours.

4.2 Accuracy on word wise prediction



Observation: Again CRF gives us a better result than SVM-MC as it takes into context of its neighbours and above all even when words are tampered the accuracy drops only slightly. On

the other hand, SVM-MC does poorly as it doesn't take into context of its neighbours and when tampered the SVM-MC's accuracy is below 1% which is not at all satisfactory.