Wanze Xie
A92061040
Cogs 185 - HW2

* Code of this assignment is attached to a zip file along with the submission

1. Task 1: Most Violating Constraints

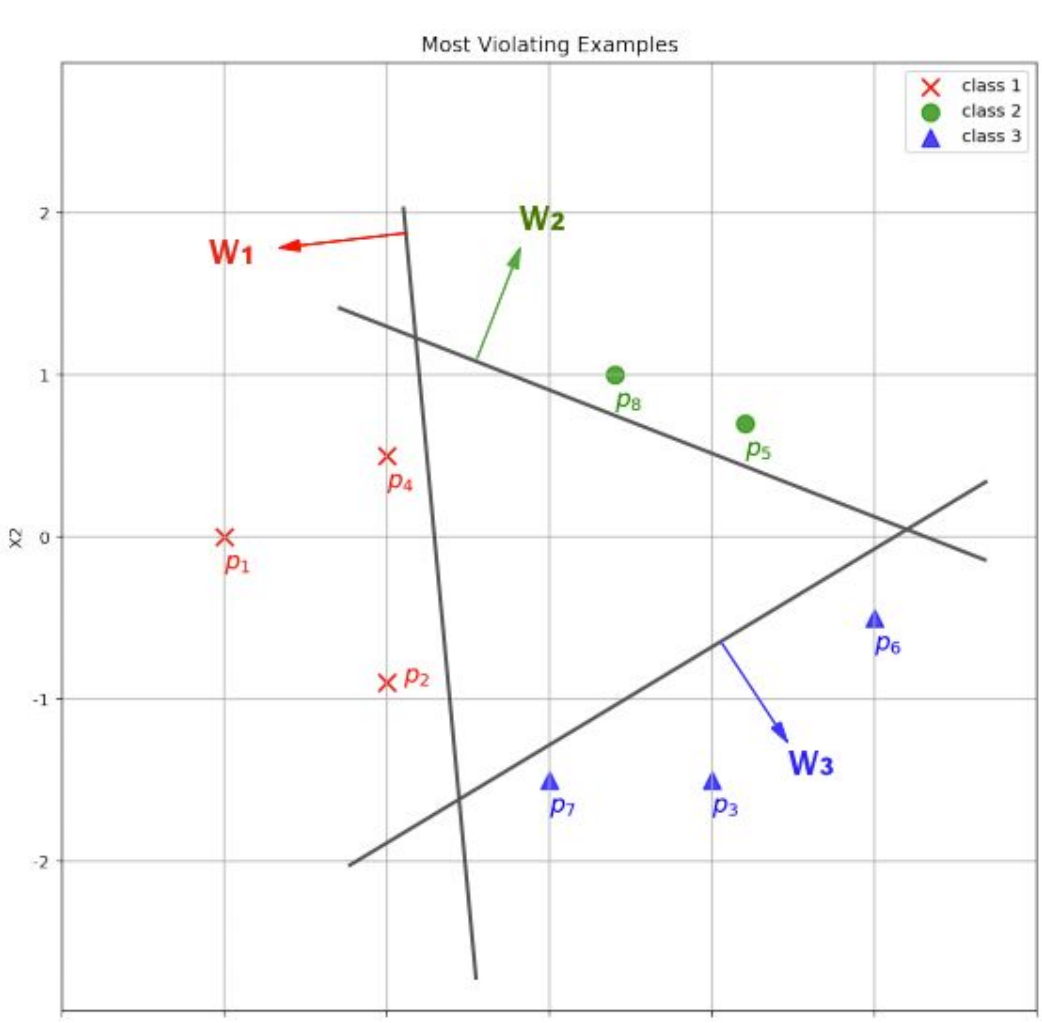
(1) The list of all examples in every "Obtain S" Step

After computation, we find that the x_2, x_6, x_8 are the violating examples, and the obtained working set S is as following

$$S = \{ (x_1, 1, 1),\ (x_2, 1, 3),\ (x_3, 3, 3),\ (x_4, 1, 1),\ (x_5, 2, 2),\ (x_6, 3, 2),\ (x_7, 3, 3),\ (x_8, 2, 1) \}$$

(2) Draw by hand the three decision boundaries after simulation in figure 2.

The figure is as shown below. It is not accurate since not required, but it is certain to have no violating constraint in the trained SVM. Drawing with computer is adopted because the result is clearer and cleaner.



2. Task 2: Structured Learning

(1) Simulate new feature using one-hot Encoding

#	Letter	Code (X_i)	Code (y_i)
1	a	$(x_i, 0, 0, 0)$	1,0,0,0
2	t	$(0, x_i, 0, 0)$	0,1,0,0
3	c	$(0, 0, x_i, 0)$	0,0,1,0
4	g	$(0, 0, 0, x_i)$	0,0,0,1

The simulated new feature is as following

$$\phi_{oh}(X_i, y_i) =$$

$$((-2.0,-1.1,0,0,0,0,0,0,0,\ 1,0,0,0,\ 0,0,0,-1.3,0,0,0,0,\ 0,1,0,0,\ 0,0,0,0,3.1,0.3,0,0,\ 0,0,1,0,\ 0,0,0,0,0,0,3.6,-1.1,\ 0,0,0,1),$$
$$(-1.0,0.2,0,0,0,0,0,0,0,\ 1,0,0,0,\ 0,0,0,0,-0.9,-1.9,0,0,\ 0,0,1,0,\ 2.6,1.5,0,0,0,0,0,0,\ 1,0,0,0,\ -1.3,-1.3,0,0,0,0,0,0,\ 0,0,0,1),$$
$$(0,0,0,0,0,0,1.0,0.4,\ 0,0,0,1,\ -1.5,-0.3,0,0,0,0,0,0,\ 1,0,0,0,\ 3.0,-0.2,0,0,0,0,0,0,1,0,0,0,\ -0.3,-0.2,,0,0,0,0,0,0,\ 0,0,0,1),$$
$$(0.4,-1.2,0,0,0,0,0,0,0,\ 1,0,0,0,\ 0,0,1,-1.2,0,0,0,0,\ 0,1,0,0,\ 0,0,0,0,2,0,0,0,\ 0,0,1,0,\ 0,0,0,0,0,0,1.2,-0.9,\ 0,0,0,1))$$

(2) Simulate new feature based on changing numbers using sliding window of size 2.

#	Code (X_i)	Slide window	Code
1	$(x_i, 0, 0, 0)$	$y_1 \neq y_2,\quad y_2 \neq y_3,\quad y_3 \neq y_4$	1,1,1
2	$(0, x_i, 0, 0)$	$y_1 \neq y_2,\quad y_2 \neq y_3,\quad y_3 = y_4$	1,1,0
3	$(0, 0, x_i, 0)$	$y_1 \neq y_2,\quad y_2 = y_3,\quad y_3 = y_4$	1,0,0
4	$(0, 0, 0, x_i)$	$y_1 \neq y_2,\quad y_2 \neq y_3,\quad y_3 \neq y_4$	1,1,1

The simulated new feature is as following

$$\phi_{oh}(X_i, y_i) =$$

$$((-2.0,-1.1,0,0,0,0,0,0,0,\ 0,0,0,-1.3,0,0,0,0,\ 0,0,0,0,3.1,0.3,0,0,\ 0,0,0,0,0,0,3.6,-1.1,\ 1,1,1),$$
$$(-1.0,0.2,0,0,0,0,0,0,0,\ 0,0,0,0,-0.9,-1.9,0,0,\ 2.6,1.5,0,0,0,0,0,0,\ -1.3,-1.3,0,0,0,0,0,0,\ 1,1,0),$$
$$(0,0,0,0,0,0,1.0,0.4,\ -1.5,-0.3,0,0,0,0,0,0,\ 3.0,-0.2,0,0,0,0,0,0,\ -0.3,-0.2,,0,0,0,0,0,0,\ 1,0,0),$$
$$(0.4,-1.2,0,0,0,0,0,0,0,\ 0,0,1,-1.2,0,0,0,0,\ 0,0,0,0,2,0,0,0,\ 0,0,0,0,0,0,1.2,-0.9,\ 1,1,1))$$

In the simulation, if the adjacent label are different, then it will be encoded as 1, otherwise 0. Since the window size is 2 and in our definition we only compares, y_1 and y_2, y_2 and y_3, y_3 and y_4, the resultant coded label should have 3 digits. This strategy reduced the spaces compared to the one-hot encoding strategy especially when the number of labels is huge

3. Task 3: Structured SVM

(1) A short description of the library I used:

The library I used for training the Structured SVM for the OCR dataset the is dlib library, where the dlib library is a machine learning library implemented in C++. Since I coded purely in python without using conda or jupyter notebook. I installed the dlib using the command `$pip install dlib .`

It is worth noting that I am using python 2.7 and I referred to some sample code snippet provided at course page. When using the dlib, I specifically implemented the *`make_psi`* function according the documentation page of the dlib and the *`separation_oracle`* function, where the first function is used by separation_oracle and during the training/testing accuracy determination stage to compute the psi(x, labels) and the second function is used by dlib's powerful function named *`structural_svm_problem`* to decide the most violating constraint, where the same strategy is used during the testing accuracy and training accuracy determination stage.

(2) The definition of my psi function:

The way I define my psi function is to take in the (features,label) from my training sample and return the psi value to the client. Instead of using one-hot encoding to simulate new features, I am using the changing numbers strategy to encode. The detailed description is as following, which is also discussed in the report part of this assignment.

In general, the dimension of my psi is calculated using the following formula:

$$(\#\ of\ features\ +\ 1)\ *\ (\#\ of\ labels)\ *\ (\#\ of\ letters\ per\ sample\ /\ window\ length)\ +\ (\#\ bits\ of\ encoded\ y)$$

And the resultant dimension is thus:

$$(128\ +\ 1)\ *\ (26\ +\ 1)\ *\ 2\ +\ 1\ \ =\ 6967$$

Note:

a. # of features are 128 as given in the dataset, plus 1 when considering the bias term
b. # of labels should be 26 for 26 characters, but I added a dummy class for the dummy padding label '_', and it is the 27th label. So the in total the labels are 27
c. The window length I used is 2.
d. The encoding strategy I used is by considering "changing labels", so if the adjacent labels are different, the encoded value is 1, otherwise 0. But either way, it will just be of 1 digit

The implementation of my psi function in python can be found in my uploaded file "task3.py"

(3) A table of training and test accuracy:

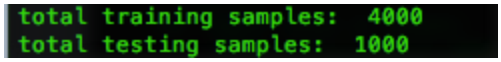
Due to the large overhead in training and the time constraint, only the window size of 2 is reported here. There were attempts for training with window size of 3 and 5, but the incredible training time makes the result unable to come out within couple hours.

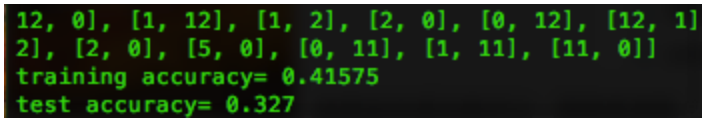
Therefore, the resultant training accuracy and testing accuracy with a window size of 2 in this report is shown as following:

Training sample	Testing sample	Window size	Training accuracy	Test accuracy
4000	1000	2	0.4158 (~41.6%)	0.3270 (~32.7%)

Note: my average training time for window size of 2 is 50 minutes, and here the best training accuracy and test accuracy is reported.

Screenshots of important terminal output:





4. Task 4: Conditional Random Field

(1) A short description of the library I used:

The library involved in the implementation of this part includes the pystruct for OCR data loading, sklearn.svm for training using LinearSVC, pycrfsuite for implementing conditional random field and training using the CRF model.

The most essential library for training is the python-crfsuite, and since the implementation is done purely in python, and I did not use conda or ipython notebook, I installed the library using `$pip install python-crfsuite`, this library is basically a python binding to CRFsuite, which is an implementation of conditional random field for labeling sequential data. Here in this problem, I am running the CRF model on the OCR dataset to predict the sequence of labels for the corresponding sequences of the input samples.

The detailed implementation of this part can be found in the uploaded file “task4.py”

(2) Use pystruct library to load the OCR model:

The code snippet in the implementation of loading the OCR model is attached below:

```
import numpy as np
from sklearn.svm import LinearSVC
from pystruct.datasets import load_letters
letters = load_letters()
X, y, folds = letters['data'], letters['labels'], letters['folds']
# we convert the lists to object arrays, as that makes slicing much more
# convenient
X, y = np.array(X), np.array(y)
```

Note that loading it in this way helps processing the raw dataset and converting the raw letters into words. Since the window length is two in the following implementation, the two letters can come from different positions in the word.

(3) & (4) Use pyscfsuite to train a CRF model and configure Trainer in pycrfsuite:

The core code snippet in the implementation of this part is attached below.

```
import pycrfsuite
trainer = pycrfsuite.Trainer(verbose=True)

X_train, X_test = X_features[folds == 1], X_features[folds != 1]
y_train, y_test = y[folds == 1], y[folds != 1]
y_tr = []
for y_i in y_train:
    z = []
    for i in y_i:
        z.append(str(i))
    y_tr.append(z)

y_te = []
for y_i in y_test:
    z = []
    for i in y_i:
        z.append(str(i))
    y_te.append(z)
cnt=0
a=0
for xseq, yseq in zip(X_train, y_tr):
#     print(xseq)
#     print(yseq)
    cnt +=1
    ystr = [str(i) for i in yseq]
    if(len(ystr)!=len(xseq)):
       print(cnt)
       continue
    a+=1
    trainer.append(xseq, ystr)
print(a)
print(len(y_train))
trainer.set_params({
    'c1': 0.10,
    'c2': 1e-3,
    'max_iterations': 100,
    'feature.possible_transitions': True
})
trainer.train('ocr.crfsuite')
```

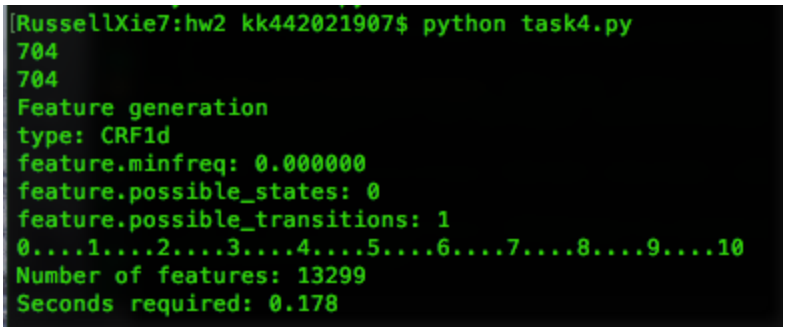
It is worth noting that I experimented with different max_iterations and different window sizes, and since the result looks no much difference from setting the max_iteration to 60 and window size to 2, the result is not reported here, but is discussed in the report part.

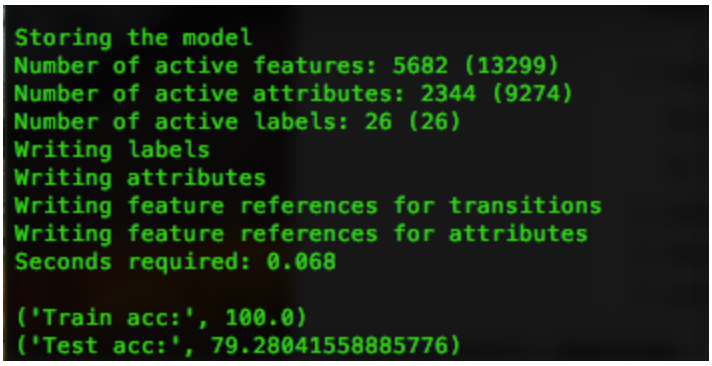
(5) Report basic training and test accuracies with window size 2.

The basic training accuracy and testing accuracy with window size 2 is reported as shown below:

Dataset	Window Size	Training accuracy	Testing accuracy
OCR	2	1.0 (~99%-100%)	0.7928 (~79.3%)

The screenshots of important terminal output:





5. Task 5: Auto-context

The experiment with Auto-context involves using reference in following url:
http:// pages.ucsd.edu/~ztu/publication/fixed-point_mac.zip

6. Task 6: Required Brief Report

6.1 Introduction:

In this assignment I primarily explored the implementation of Structured SVM and the implementation of Conditional Random Field model. During experimentation, the focus is on how the window size might affect the training result. The dataset used is OCR dataset that contains around 6,000 handwritten words, with average length of 8 characters. During implementation of structured SVM, the dataset is refactored and decomposed to training set of 4,000 samples and testing set of 1,000 samples. And in the implementation of the CRF model, all the dataset is loaded before processing the data.

6.2 Methods and Experiment:

6.2.1 Structured SVM

In the first two part of the question, I was primarily presenting the algorithm and high level concept of implementing finding the most violent constraint and computing psi for structured svm training. And the idea of the first two part of the assignment is fully implemented in the third part, where the make_psi function and separation_oracle function is used. The implementation of Structured SVM is based on the dlib library [2] as described in the third part of the assignment, but the data loading and data processing are processed with consideration and care.

When refactoring the OCR data [1], I was continue reading the words from the data until I reached 4001 characters for training and 1001 for testing. During concatenation, I added a special character '_' in between words. So for example if the word is apple and banana, my sample sequence would be ap, pp, pl, le, e_, _b, ba, an, na, an, na. This strategy is used for both the training sample and testing sample, and each two character composes a sample, where each element/character in the sample is composed of 128 features. The total number of labels are 27 because there are 26 different characters in total and the 1 label for the special character '_' that is placed in between the words.

During experimentation, I discovered that the training accuracy and testing accuracy are both not very ideal. With a window size of 2, the training accuracy I obtained is around 41.6% and the test accuracy obtained is around 32.7%. One discovery I made is that since I am using the first 5,000 characters to compose the training and testing sample, the word of choice is actually pretty limited, where only "mmanding", "brace", "lcanic", "comfortably" and "vving" this 5 different type of words showed up in the sample, but the features in the sample are different for different handwriting words. Because of this limitation, it makes sense that the training accuracy and testing accuracy is not very ideal.

With experiments, it is discovered that adding some randomness when preprocessing the training sample and testing sample helps improve overall accuracy, although it is considered that the dlib library already introduced randomness to the sample data during the training, so the as expected, the performance improvement is not very obvious. Also it is yet to experiment with larger window size and different encoding strategy to do the experimentation. It is worth noting that when counting

accuracy, I chose the version 2 as noted in the piazza post and I compared each single letter with the label to count the error and compute accuracy.

6.2.2 Conditional Random Field

The implementation of Conditional Random Field is primarily based on the reference from the github site of pystruct [3] and the crfsuite documentation [4]. There are some parts that is edited and implemented by me in order to do the experimentation of the CRF model on the OCR data. The detail is discussed in the following part.

During experimentation I tempted to train the CRF model by specifying different window size and different max iterations. But the training accuracy is already very high with the max iteration of 60 and window size of 2, and the accuracy is 99% - 100% already so the changing of parameters does not make much sense and is not necessary for improving the training accuracy. The testing accuracy is high as well with an accuracy of nearly 80%, which is way higher than the training result obtained using the structured SVM. However, it is discovered that when I changing the window size to 3, I discovered that the obtained accuracy is slightly higher, but not very obvious. It is believed that the larger the window size the better the training result should be, but here because I only experimented with a window size of 3 I believe it is too soon to make the conclusion. Also, due to the time constraint, I did not have chance to experiment with more different kind of window sizes. In terms of the max iteration, it looks like 60 is pretty decent already, and other values does not seem to change the result obviously as well, the value I experimented includes 80, 100 and 200. As a result, the report of the training result of window size of 2 is provided because window size of 3 does not change the performance significantly.

6.3 Discussion and Summary:

In this study and report, I have been discussed the design of the structured SVM and the CRF model using the OCR dataset as sample. It is also discussed the situations under which we might obtain higher performance than the other. One of the core factor that would affect the training result is the window size or window length when we are processing the data sample, and intuitively the larger the window size, the training result tends to be better, but so far it lacks evidence to say it for sure. We did this experiment in CRF model but not in structured SVM model primarily because the structured SVM model consumes too much time, and each full scale training will require at least around an hour to finish.

As a brief summary, it shows that the structured SVM does not perform very well on the OCR data, and it is likely that changing the window size and adding more randomness might improve the result. The CRF model is more preferred because the obtained result is much better. In terms of selecting which model for training, it is believed that more experiments are still needed to reach a conclusion and these two models are all worthwhile to try out if time constraint allows.

6.4 Reference:

[1] OCR datasets: http://ai.stanford.edu/~btaskar/ocr/

[2] Referenced snippet for structured SVM algorithm: http://dlib.net/svm_struct.py.html

[3] Pystruct github: https://pystruct.github.io/user_guide.html#chaincrf

[4] Python-crfsuite documentation https://python-crfsuite.readthedocs.io/en/latest/