# Comparing Various Frameworks for Structured Prediction on OCR

**Rebecca Du**
(rrdu@ucsd.edu)

## Abstract

This paper explores the performance of different frameworks on the structured prediction problem. 3 different frameworks (Auto-Context, CRF, and Structured SVM) were tested on the OCR dataset. For each of the frameworks, different window sizes as well as training and testing sample size splits were used for a thorough investigation. Results indicate that the CRF frame work performed the best, and that the best window size in general is a window size of 2, and the best training/testing data split is 4,000/1,000.

## 1. Introduction

In this final project, the OCR (Optical Character Recognition) dataset was used for the structured prediction task. The dataset is included in the Github repository linked at the end of this section.

The code for the final project (including loading, formatting, processing, and classifying the datasets) was written in the Python programming language (https://www.python.org/). Various Python libraries were used to assist the coding process, including Numpy (http://www.numpy.org/) and Matplotlib (https://matplotlib.org/). The code was written in Jupyter notebooks (https://jupyter.org/).

The general framework for the different models (Auto-Context, CRF, Structured SVM) were based on code provided in class.

The Jupyter notebooks used in this project can be found in the Github repository linked below.
https://github.com/rrdu/COGS-185-Final-Project

## 2. Methodology

### 2.1 General Approach

The overall goal of this project was to assess how varying the window size and training and testing sample size splits would impact the performance of various frameworks on the structured prediction task for the OCR dataset.

The OCR (Optical Character Recognition) dataset that was used for this experiment was subsidized into 5,000 words for the sake of training and testing efficiency (otherwise the process would have taken too long). The dataset has 26 labels for each letter of the alphabet.

The goal of the structured prediction task is to train a given framework with the OCR data so that it will eventually be able to correctly identify the letter in the image. In this experiment, 3 frameworks were adopted: **Auto-Context**, **CRF**, and **Structured SVM**. The frameworks and implementations are explained in depth in the following sections.

### 2.2 Auto-Context

Auto-context refers to a method of improving model performance by using the context of the surrounding information in the prediction process. In the context of the structured prediction task at hand, it involves leveraging information about neighboring characters to improve the recognition accuracy of the current character. The auto-context method uses multiple iterations, and in each iteration the model makes predictions which are then used as additional features (aka context) for the next iteration. This enables the model to refine its predictions after every iteration by incorporating increasingly accurate contextual information. Additionally, the augmented features provide a better representation that includes both the original input data and contextual information, which helps the model learn relationships that the original features cannot capture alone.

In this experiment, auto-context was applied to a structured SVM framework to improve its performance. The specifics of the structured SVM framework itself can be found in section 2.4.

Auto-context was implemented on the structured SVM framework through the creation of an `AutoContextProblem` class that defines a `make_psi` method which adds context into feature vectors and updates the context based on predictions. The context is initialized as a zero matrix with the same dimensions as the sample size and window length. The `make_psi` method then constructs the feature vector for each sample by combining both the raw features and context features. The context features are then updated based on predictions from previous iterations.

### 2.3 CRF

CRF, or Conditional Random Fields, are a type of statistical modeling method that are effective at sequence labeling tasks like part-of-speech tagging and OCR. Specifically, they are undirected graphical models (aka Markov Random Fields) that use a graphical representation of conditional probabilities given input features. The nodes in the graph represent observed and hidden variables while the edges represent the dependencies between the variables.

A key difference between CRFs and Hidden Markov Models (HMMs) arises from how they model probability. CRFs model the conditional probability $P(Y \mid X)$ of a sequence of labels $Y$ when given a sequence of observations $X$. HMMs, on the other hand, model the joint probability $P(Y, X)$. Since CRFs are log-linear models, the conditional probability is represented as a normalized exponential function of the sum of the weighted feature functions.

CRFs use feature functions to capture the relationship between the input data and output labels as well as between adjacent output labels. The training process involves finding the optimal weights for these feature functions through gradient-based optimization methods, with the goal being maximizing the likelihood of the observed label sequence that the input has.

In this experiment, CRF was implemented through the `sklearn_crfsuite` library. The `convert_features` function converts raw feature sequences into a format more suitable for CRF: a list of dictionaries where each dictionary represents the features of a single character. The `convert_labels` function similarly transforms labels into a format more suitable for CRF by turning them into a list of string labels.

The CRF model is created using the `sklearn_crfsuite.CRF` class before being trained using the pre-built `fit` method.

### 2.4 Structured SVM

Structured SVMs are a variant of traditional Support Vector Machines (SVMs) that are specially designed to handle structured outputs, such as sequences. While traditional SVMs predict a single label for each output, structured SVMs can predict complex outputs with interdependent components, such as in OCR where the output is a sequence of characters rather than just one character. The dlib library - a C++-implemented machine learning library with a Python interface - was used in the implementation of structured SVM.

Structured SVMs use joint feature maps $\psi(x,y)$ that encode the relationship between the input $x$ and structured output $y$. The model learns a weight vector $w$ such that the score $w \cdot \psi(x,y)$ is high for correct outputs and low for incorrect outputs. This is implemented in the code through the `make_psi` function, which defines a feature vector ψ that is structured according to a pattern based on the input feature and corresponding label.

During the training process for structured SVMs, loss-augmented inference is performed in order to find the most-violating constraint. Essentially, this involves finding the output which maximizes the sum of the model score and the penalty from predicting the output instead of the true output.

The key parts of the code that implement structured SVM are the `ThreeClassProblem` class, the make_psi function, and the `separation_oracle` method. The `ThreeClassProblem` contains the `make_psi` function and the `separation_oracle method`, as well as a function for calculating training and testing accuracies. As previously mentioned, the `make_psi` method creates the joint feature vector given an input *x* and an output *label*. The feature vector includes a bias term as well as features for each label in the sequence. The `separation_oracle` method performs the aforementioned loss-augmented inference by iterating over possible labels to find the one which maximizes the sum of the model score and the loss. It uses the current weight vector to evaluate different configurations and choose the best one. The structured SVM model is then trained using `dlib.solve_structural_svm_problem`.

### 2.5 Window-Size and Training/Testing Split Variations

For each framework, 2 window sizes were tested: a window size of 2 characters and 3 characters.

Additionally, 3 different training and testing sample size splits were used for each framework:

- 4,000 training and 1,000 testing
- 2,500 training and 2,500 testing
- 1,000 training and 4,000 testing

For each implementation, the final training and testing accuracies were printed out at the end. The weights obtained from the frameworks were saved and are available on the Github repository.

## 3. Results

The training and testing accuracies for each of the frameworks and the various window sizes and training/testing splits are listed in Table 1 in the appendix.

The results indicate that for auto-context, the best performance was obtained with window size 2 and a 2,500/2,500 training/testing split for a 45.5% testing accuracy.

For CRF, the best performance was obtained from window size 3 with a 1,000/4,000 training/testing split that resulted in 88.6% testing accuracy.

For Structured SVM, the best performance was obtained from window size 2 with a 2,500/2,500 training/testing split that resulted in 44.6% testing accuracy.

## 4. Discussion

The results will be analyzed first within the context of each framework. Then, overall trends will be discussed.

For auto-context using a window size of 2, the results indicate that as the training sample size increases, the training accuracy decreases. This could be due to the model overfitting smaller training sets but struggling to generalize when more data is introduced. The trend is amplified with a window size of 3, indicating that increasing the window size increases the complexity of the task, which worsens model performance. Across both window sizes, auto-context suffers from low testing accuracy. This may be due to a lack of contextual information, which may cause incorrect initial predictions/features that are further propagated through the iterative process. Overall, the results show that auto-context is less suitable for the OCR task than algorithms like CRF, which are specifically designed to handle sequential data. The implementation could be improved with further hyperparameter tuning (such as varying the values for Niter) and more enhanced feature representation.

The CRF implementation saw high training accuracy scores across both window sizes 2 and 3, indicating that the model is very effective at fitting the training data. Additionally, both window sizes demonstrated high testing accuracy as well, though a window size of 3 was slightly preferred due to outputting a higher testing accuracy. One trend of note in the results is that the testing accuracy decreased slightly with larger training sets, possibly due to overfitting or a need for more regularization. CRF's good performance with a window size of 3 indicates that the model can effectively leverage the increased context it gains from the expanded window size in a way that the other models cannot.

Finally, in the Structured SVM implementation with window size 2, there was great variety in training accuracy, indicating instability. Additionally, the low test accuracy suggests that structured SVM may not be as effective as CRF when generalizing to unseen data. For window size 3, the training and testing accuracies were even lower, indicating the structured SVM struggled with the increased complexity that comes from a larger window size. The ~10% drop in testing accuracy indicates that the implementation may need further fine-tuning to handle larger window sizes effectively.

Overall, CRF stood out as the best framework for the structured prediction task on the OCR dataset. Its consistently high training and testing accuracies across the different window sizes and training/testing splits speaks to its ability to accurately model dependencies between adjacent labels. It was the only framework that saw an increase in testing accuracy with a

window size of 3, which otherwise hindered the performance of auto-context and structured SVM due to their inability to leverage the added context in the same way.

## 5. Conclusion

This paper investigated the influence of varying window sizes and training/testing sample size splits for different frameworks tasked with completing the structured prediction problem on the OCR dataset. The results indicated that the CRF framework performed the best on the task. Though the CRF framework performed best on a window size of 3 and with a 1,000/4,000 data split, in general the best settings are a window size of 2 and a 4,000/1,000 data split that allows the model to thoroughly train on data before being put to the test.

The overall trend for auto-context and structured SVM seemed to be that as the window size increased, the testing accuracies decreased due to the increased complexity. Additionally, greater training sample sizes generally resulted in higher testing accuracies. The only exception to this was the CRF framework, which had high accuracies across the board for the various window sizes and data splits. This means that it is able to generalize accurately even with minimal training sample size. CRF's good performance can be explained due to the model being designed to handle sequential data that has dependencies between the labels (like OCR). Its ability to extract features like pixel intensities and edge detections, as well as its optimization of parameters to maximize the probability of the correct label sequences make it ideal for solving the structured prediction problem. Future research should aim to improve the performance of auto-context and structured SVM through trying other implementations as well as adjusting hyperparameters to see if it improves overall performance.

# A. Appendix

## A.1. Tables

*Table 1: Training and Testing Accuracies across Window Sizes and Training/Testing Splits*

**Note*: the best testing accuracy for each window size is highlighted, while the best testing accuracy for the framework regardless of window size and testing split is **bolded**.*

| Framework | Window Size | Training Sample Size | Testing Sample Size | Training Accuracy | Testing Accuracy |
|---|---|---|---|---|---|
| **Auto-Context** | **2** | 1,000 | 4,000 | 53.2% | 10.5% |
| | | **2,500** | **2,500** | **49.1%** | **45.5%** |
| | | 4,000 | 1,000 | 49.5% | 43.5% |
| | **3** | 1,000 | 4,000 | 37.6% | 6.5% |
| | | 2,500 | 2,500 | 33.7% | 27.9% |
| | | 4,000 | 1,000 | 32.7% | 29.2% |
| **CRF** | **2** | 1,000 | 4,000 | 97.1% | 86.7% |
| | | 2,500 | 2,500 | 98.2% | 85.5% |
| | | 4,000 | 1,000 | 99.7% | 81.6% |
| | **3** | **1,000** | **4,000** | **96.9%** | **88.6%** |
| | | 2,500 | 2,500 | 98.6% | 86.3% |
| | | 4,000 | 1,000 | 99.7% | 82.8% |
| **Structured SVM** | **2** | 1,000 | 4,000 | 54.8% | 39.5% |
| | | **2,500** | **2,500** | **50.6%** | **44.6%** |
| | | 4,000 | 1,000 | 49.1% | 44.1% |
| | **3** | 1,000 | 4,000 | 54% | 41.7% |
| | | 2,500 | 2,500 | 33.7% | 27.5% |
| | | 4,000 | 1,000 | 32.1% | 29.4% |