Old Bailey Decision Classifier

Muteeb Akram Nawaz Doctor[1*]

[1]Kahlert School of Computing, The University of Utah

Abstract

The "Old Bailey Decision Classifier" is a machine learning project to predict the outcomes of criminal trials held at the Old Bailey, the central criminal court of England and Wales. Utilizing a diverse dataset of trial transcripts, case summaries, and others. Different machine learning algorithms and various classifiers are employed to predict sentencing decisions. The classifier achieves high accuracy in predicting verdicts, such as guilty or not guilty, through dataset cleaning and transformation, model optimization, and fine-tuning hyperparameters. The Old Bailey Decision Classifier describes authors understanding of Machine Learning algorithms and their implementation to solve a real-world problem using different classifiers and algorithms.

Introduction

An Overview

The Old Bailey is the colloquial name for the Central Criminal Court of England and Wales, handling significant criminal cases in London and occasionally from across England and Wales. It has been around since the 16th century. The hearings from courts have been digitized and published online via the Old Bailey Proceedings Online project (https://www.oldbaileyonline.org/).

Since all the text of the trials from 1674 to 1913 is available, we can ask the following text classification question: *Can we predict the decision of the court using the transcribed dialogue during a trial?* I intend to explore Machiner Learning algorithms and classifiers that predict the outcomes of trials. That is, the instances for classification are the transcripts of trials, and the labels are either guilty (denoted by 0) or not guilty (denoted by 1).

Datasets

The raw text from the hearings is pre-processed, and different feature representations of trials are used to help classify.

- **bag-of-words**: The bag of words representation represents the text of the trial as a set of its words, with their counts. That is, each dimension (i.e feature) corresponds to a word, and the value of the feature is the number of times the word occurs in the trial. To avoid the dimensionality from becoming too large, the features are restricted to use the 10,000 most frequent words in the data.

- **tfidf**: The tfidf representation is short for term frequency–inverse document frequency, which is a popular document representation that seeks to improve on the bag of words rep- resentation by weighting each feature in the bag-of-words representation such that frequent words are weighted lower. As with the bag-of-words, the features are restricted to use the 10,000 most frequent words in the data.

- **glove**: The previous two feature representations are extremely sparse vectors, with only a small fraction of the 10,000 features being non-zero for any vector. The glove representation represents a document by the average of its "word embeddings", which are vectors that are trained to capture a word's meaning. The word embeddings are dense, 300 dimensional vectors, and for the purpose of this project, each word is weighted by its tfidf score.

- **misc**: In addition to these features, miscellaneous categorical attributes are extracted from the trial data. The categories before cleaning the dataset are represented in Table 1.

Table 1: *Misc Dataset Features.*

Features
defendant age
defendant gender
num victims
victim genders
offence category
offence subcategory

*Corresponding author: muteebakram@gmail.com
Submitted: April 30, 2024; **UID:** u1471482

Methodologies

Ideas Explored

Decision Trees

The misc dataset consists of attributes and attribute values representing real-world features as strings. Thereby, using a decision tree made the right choice for me. The hypothesis space would be all possible decision trees, and the learning algorithm I chose was ID3 with information gain as entropy.

Preprocessing of Misc dataset:

Looking closely into the training dataset of misc, it was clear that feature values were inconsistent, so I did the following for this dataset.

- The label column was missing. Since it was mentioned, every dataset represented the same example. I used the glove dataset label column to train, test, and eval datasets of misc, respectively.
- The defendant's age was an integer and sometimes a descriptive string. Therefore, printed all the unique values of the misc dataset's six attributes to identify if other attributes also required cleaning. Only the defendant age had mixed data types. I used the pandas replace feature with the dictionary to replace all the strings with integer strings.
- Converted all the int64 data types to string for a consistent data structure.

Implementation:

As previously mentioned, I used ID3 algorithms, using entropy as information gain, to determine the feature that most effectively classifies the remaining attributes. I implemented both *full trees* and *depth-limited trees*, representing the Decision Trees as Python Dictionaries.

Additionally, a *tree_walk* function was created to recursively traverse the trees recursively and reach the leaf nodes for label prediction. In the case of depth-limited trees, the depth of the decision tree serves as a hyperparameter. Consequently, I initially determined the optimal hyperparameter and subsequently retrained the model with the best depth setting.

The results of the decision tree is represented in Table 2.

Results:

Table 2: *Decision Tree Results Accuracy.*

Full-Tree			Depth-Limited Tree ($d = 6$)		
Train	Test	Eval	Train	Test	Eval
79.825%	72.8%	73%	79.82%	67.23%	72%

Perceptron

I implemented a mistake-bound online learning setup for the perceptron, updating weights only when a mistake was made. Variants including simple, average, and aggressive perceptron with learning rates (η) and margin (μ) of [1, 0.1, 0.01, 0.001] were tested over 20 and 50 epochs, though little difference was observed beyond 20 epochs. Dataset labels, originally 0 or 1, were modified to -1 and 1 to fit the perceptron algorithms. Initial feature weights ranged from -0.01 to 0.01, with a bias weight of 1.

One of the challenges I faced was the low accuracy of perceptron, less than 50%, with labels 0 and 1. It worked after debugging and changing the labels to -1 and 1, respectively. Later, I replaced -1 with 0 for the eval submission. Another improvement of 2% accuracy comes by shuffling the training dataset every epoch with the same random seed.

Simple Perceptron: An update will be performed on an example (x, y) if $y(w^T x + b) < 0$ as:

$$w \leftarrow w + \eta y x,$$
$$b \leftarrow b + \eta y.$$

Average Perceptron: In addition to the original parameters (w, b), you will need to update the averaged weight vector and the averaged bias b_a as:

$$a \leftarrow a + w$$
$$b_a \leftarrow b_a + b$$

Aggressive Perceptron: Unlike the standard Perceptron algorithm, here the learning rate η and update only within μ is given by

If $y(w^T x) \leq \mu$ then Update $w_{new} \leftarrow w_{old} + \eta y x$

$$\eta = \frac{\mu - y(w^T x)}{x^T x + 1}$$

Results:

Table 3 represents the results of the simple, average, and aggressive perceptrons using the best hyperparameters for the dataset.

Table 3: *Perceptron Results Accuracy.*

Type	Test	Eval
Simple Perceptron *tfidf;* $\eta = 0.001$	72%	68%
Average Perceptron *tfidf;* $\eta = 0.1$	72.93%	66%
Aggressive Perceptron *tfidf;* $\gamma = 1$	72.8%	68%

Table 4: *SVM Test Dataset Results.*

Parameters	A	P	R	F1
TFIDF Dataset $\eta = 0.001; C = 1000$	71.77%	0.70	0.71	0.71
BOW Dataset $\eta = 0.001; C = 0.1$	69.06%	0.69	0.65	0.67
Glove Dataset $\eta = 0.001; C = 0.1$	62.71%	0.62	0.57	0.59

Support Vector Machines

I implemented the stochastic sub-gradient descent version algorithm SVM as described in the class with the learning rate for each t^{th} epoch as γ_t and objective function as J. The algorithm to update the weights is mentioned below.

$$\gamma_t = \frac{\gamma_0}{1+t}$$

$$J(w) = \sum_{i=1}^{m} \max(0, 1 - y_i w^T x_i) + \frac{1}{2} w^T w$$

Algorithm 1 Sub-Stochastic Gradient Algorithm

Input: Data set D, learning rate η, number of epochs T
Output: Optimized parameter vector $\mathbf{w}$
Initialize parameter vector $\mathbf{w} = \mathbf{0}$
for $t = 1$ to T **do**
  Randomly select a subset S_t of the data set D
  for all $(\mathbf{x}_i, y_i) \in S_t$ **do**
    Compute the sub-gradient $\nabla J(\mathbf{w})$ with respect to the loss function
    Update parameter vector: $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla J(\mathbf{w})$
  end for
end for

Stopping Strategy:

The training process halts either when the absolute relative change in the objective function falls below a specified threshold or when 50 epochs have been completed, whichever condition is met first.

Since the datasets are large, convergence might not happen. Thus, 50 epochs limit is used, which also helps to avoid over-training.

Results:

Table 4 represents the Support Vector Machines (SVM) results with the best hyperparameters setting for the tfidf, bow, and glove test datasets.

Figure 1 represents the SVM's loss at each epoch during training for the tfidf, bow, and glove train datasets with best hyperparameters setting.

Ensemble

I implemented a majority ensemble method that selects the most commonly predicted or voted label from the weak classifiers. In the event of a tie among all weak classifiers, a random label is chosen. These weak classifiers are selected based on their highest accuracy achieved in previous milestones, with their best hyperparameter settings. The members of this ensemble and the weak classifiers are:

1. SVM Bag Of Words Dataset
2. SVM Glove Dataset
3. SVM TFIDF Dataset
4. Decision Tree Misc Dataset
5. Simple Perceptron TFIDF Dataset
6. Aggressive Perceptron TFIDF Dataset

The weak classifiers are trained on mixed datasets, yielding a multi-dataset and multi-algorithm classifier ensemble. One challenge I encountered in implementing this ensemble was preserving the weights or trees of the best classifiers for subsequent use in making predictions on test and evaluation datasets. Although I successfully utilized numpy's save and load functions to preserve and recover weights, the weights were stored in different data types, and I could not perform mathematical operations on these weights.

To mitigate this issue, I planned to reuse the eval dataset predictions from the previous milestones. Leveraging the predictions from all six evaluation classifiers, I constructed a new pandas DataFrame where each row represented predictions for the same instance from every classifier as shown in the Figure 2. From this DataFrame, I generated a new prediction by selecting the most frequently occurring label and breaking the tie when necessary for each example, thereby creating a majority ensemble and achieving an accuracy of 73% on the eval dataset.

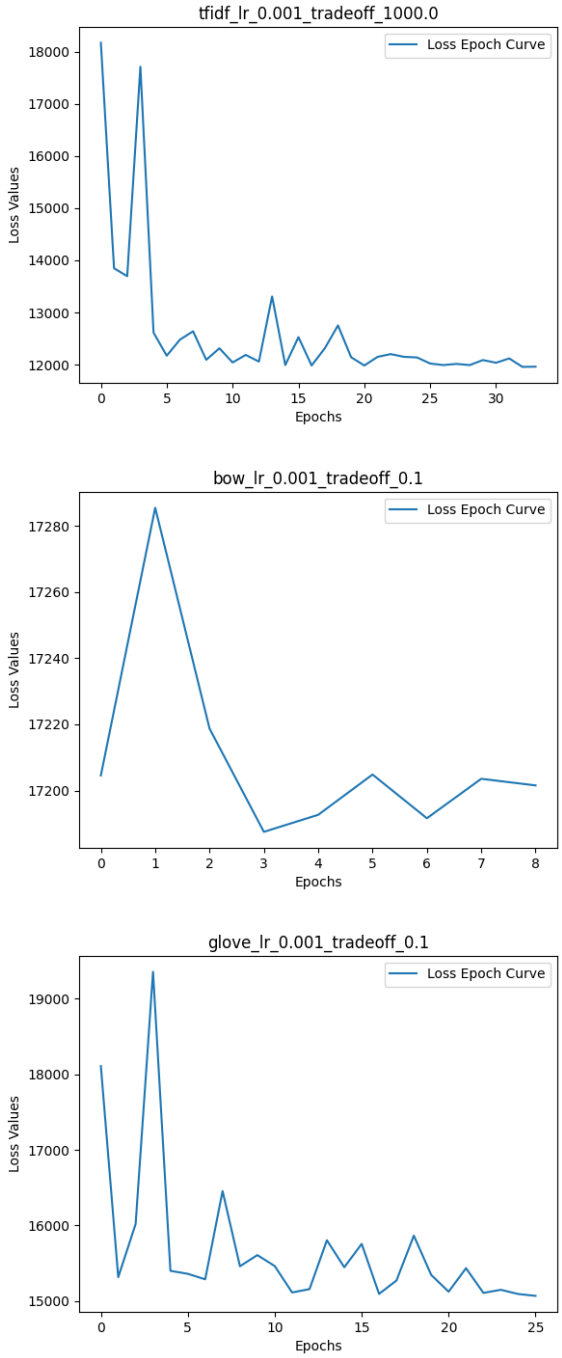

Figure 1: *SVM Training Loss Epoch Curve*

	decision_tree_labels	simple_perceptron_labels	aggressive_perceptron_labels	svm_glove_labels	svm_bow_labels	svm_tfidf_labels
0	1	1	1	1	1	1
1	1	1	1	1	1	1
2	0	0	1	0	0	0
3	1	1	1	1	1	1
4	0	1	1	1	1	1
...	...	...	...	...	...	...
5245	0	0	0	0	0	0
5246	1	1	1	1	1	1
5247	1	1	1	1	1	1
5248	1	1	1	1	1	1
5249	1	1	1	0	1	1

5250 rows × 6 columns

Figure 2: *Majority Ensemble Pandas Eval Dataset*

Neural Networks

I tested various Neural Networks using different configurations of hidden layers, loss functions, and activation functions across glove, bag-of-word, and tfidf datasets. Experimenting with increasing the size and number of hidden layers, I observed no significant improvement; instead, it only extended the training time. Interestingly, all the loss functions yielded similar results, effectively reducing the loss. Therefore, I stuck with the cross-entropy loss function.

Regarding activation functions, I explored ReLU, Sigmoid, Tanh, Leaky ReLU, and Softmax. Surprisingly, most of them yielded comparable accuracies, with Tanh slightly outperforming the others and contributing to a modest 1% increase in accuracy.

The objective function of cross-entropy loss is as follows.

$$L(y, \hat{y}) = -\frac{1}{M}\sum_{i=1}^{M}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)]$$

$$where, \hat{y}_i = y_i w^T x_i$$

The *tanh* activation is as follows.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Results:

The highest test accuracy among all the neural network configurations across datasets was achieved on the glove dataset. It consisted of two hidden layers, a dropout rate of 0.25, and used the cross-entropy loss function and the tanh activation function. The shape and test dataset accuracy of this neural network is 300x200x50x2 are 67.46% respectively. The Figure 3 represents the loss at each epoch.

Discussion

Learnings

Most of my learning came from this Machine Learning course (CS6350), my first formal learning of this domain. The well-organized course structure, assignments, syllabus, professor, and helpful teaching assistants facilitated my understanding and implementation of diverse concepts in this vast field.

The primary objective of the course was to delve into supervised learning. I learned to represent any machine learning algorithm in terms of instance spaces, label spaces, concepts, and hypothesis spaces. Thereafter, I learned various classification and regression problems, along with their respective limitations.

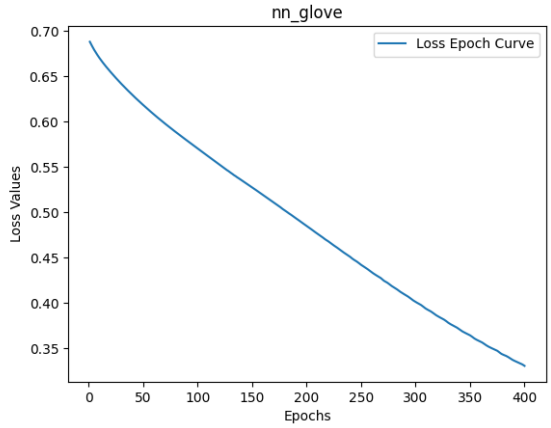

Figure 3: *Neural Network Cross Entropy Loss Epoch Curve on Glove Test Dataset*

Exploring decision trees and learning to problem as a decision tree. I delved into the intricacies of algorithms like the ID3 algorithm, designed to efficiently select optimal decisions. Subsequently, I learned about linear classifiers, understanding their significance and the problems they can effectively address.

One of the ways of learning is Mistake Bound Learning, which is updating the weights only when there is a mistake and answering the question of how many mistakes an algorithm makes before accurately predicting and studying the CON and Halving Algirthms.

Exploring the Perceptron algorithm and its connection to geometry led me to the realization that the ultimate learning objective is to enhance the fitting of linear functions for effective label classification. Various adaptations of the Perceptron, including Simple, Average, Margin, and aggressive, are different optimization of these linear functions.

Learning Theory is my favorite one. It started with how many mistakes an algorithm can make to learn the problem, which describes mistake-bound learning successfully. Then, what if I want to learn a function close to a true function but not exactly with an error and so much probability, thereby learning PAC Learning, and when is this approximately learning efficient, i.e., sample and computational complexity? Later, moving on to hypothesis space, what if we move out of our concept class and increase hypothesis space? Will we be able to generalize that better? That is Agnostic learning. Finally, as most hypothesis spaces are infinite, how can one characterize the complexity of different hypothesis spaces? Then comes the VC dimension and shattering concept, which represent the complexity and reason for which hypotheses are good. Overall, looking into the theory and effort from theory and math people being used in machine learning was fun.

Jumping onto Ensemble based learning, the key idea is that a weak group classifier can be used to create a strong classifier. Then comes the Support Vector Machines, with the idea that margin implies better generalization and treating learning problem as minimizing loss and finding the point that is the weights at which the loss is minimal using gradient descent and its variants. Loss functions are essential; different loss functions converge at different rates, and how learning rate can help.

The concept of probabilistic learning introduces a shift from discrete label values (such as 0 or 1) to using values between 0 and 1 for classification. This approach involves exploring Bayesian probability and logistic regression. The fundamental idea revolves around maximizing a Posterior (MAP), which represents the hypothesis's probability given the dataset, being directly proportional to Maximum Likelihood Estimation (MLE), which assesses how well a hypothesis performs given the dataset, along with incorporating prior knowledge of the hypothesis.

Finally, delving into multilayer neural networks, understanding the role of activation functions and different activation functions such as ReLU, Sigmoid, and Tanh. The concepts of the forward pass and backpropagation techniques for optimizing weights with the stochastic gradient descent algorithm.

Ideas From Class

- **Cross Validation**: The hyperparameters significantly affect the accuracy and the rate of learning. Thus, it is essential to use appropriate parameters for different datasets. One of the techniques is k-fold cross-validation.

  The general k-fold cross-validation approach randomly splits the training data into k equal-sized parts. Now, we will train the model on all but one part with the chosen hyper-parameter and evaluate the trained model on the remaining part. We should repeat this k times, choosing a different part for evaluation each time. This will give us k values of accuracy. Their average cross-validation accuracy gives us an idea of how good this choice of the hyper-parameter is. We will need to repeat this procedure for different choices to find the best value for the hyper-parameter. Once we find the best value of the hyper-parameter, we can use the value to retrain the classifier using the entire training set.

- **Random Seed & Hyper Parameters**: Initiating a random seed at the outset across all libraries not only ensures reproducibility but also

significantly influences the overall accuracy of the model when fine-tuning the seed value.

Aside from utilizing the default parameters, the class was instructed to experiment with additional parameters. This suggestion proved invaluable, particularly in fine-tuning the trade-off parameter, where I achieved significantly higher accuracy by adjusting a parameter not included in the default settings.

- **Stopping Strategy**: In machine learning, it is crucial to know when to stop learning to avoid overfitting and memorizing the dataset. Because overfitting gives better accuracy on the training dataset but will perform worse on the test dataset and won't generalize.

  One way to select the number of epochs is to observe the value of the SVM objective over the epochs and stop if the change in the value is smaller than some small threshold.

- **Data Preprocessing**: Improving the presentation of feature sets can significantly enhance the learning process. Even seemingly insignificant features can be transformed to enhance the quality of the dataset. Despite the inconsistencies in the misc dataset, applying preprocessing techniques outlined in the decision tree methodology enabled me to utilize it effectively, ultimately leading to the highest accuracy achieved among all submissions.

Results

The Table 5 presents my selected Kaggle submission and the Figure 4 represents the accuracy of each submission.

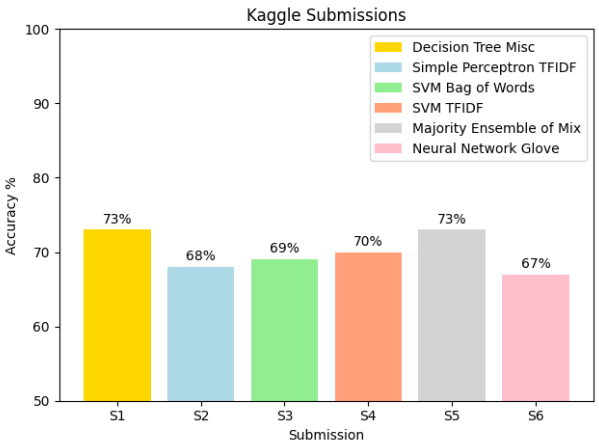

Figure 4: *Kaggle Submission Eval Dataset Accuracy*

Table 5: *Kaggle Submission*

Submission	Machine Learning Model
S1	Decision Tree Misc Dataset
S2	Simple Perceptron TFIDF Dataset
S3	SVM Bag of Words Dataset
S4	SVM TFIDF Dataset
S5	Majority Ensemble of Mix Dataset
S6	Neural Network Glove Dataset

More Time

Given more time to continue the project, I wish to work on into optimizing neural networks and exploring how various factors impact overall accuracy. Additionally, I aim to compare the performance of neural networks with other models like SVM, decision trees, and logistic regression. Feature transformation and selection are also areas of interest, as they have a significant influence on accuracy and merit further investigation.