IE 506: Machine Learning - Principles and Techniques **Jan-May 2023**

End-term Project Report : P-Ranking with Ranking

Team Name: Vasusena *Team Members: Vivek Kumar Trivedi(22N0457)*

Abstract

Ordinal regression is a type of learning that involves ranking instances with a rating between 1 and k, with the objective of accurately predicting the rank of each instance. In this project, we explore a group of closely related online algorithms for ordinal regression and evaluate their performance in terms of ranking loss. We conducted four sets of experiments on different data sets, including synthetic data and Each-Movie data sets for collaborative filtering IMDB and TMBD Movie Review Data-set. Our results show that P-Ranking outperformed other algorithms on the first two data sets, while the multi-class perceptron algorithm outperformed others on the last two data sets. However, we observed that P-Ranking only outperformed when there exists a parallel plane that separates each rank in a higher dimension. Additionally, we developed a neural network approach that provided reasonably good predictions for ordinal regression. Our findings highlight the importance of selecting an appropriate algorithm for ordinal regression, considering the distribution of data in high-dimensional space.

1 Introduction

Ranking learning is an important area of research in machine learning, with applications in fields such as information retrieval and collaborative filtering. In these tasks, the goal is to assign a rank or rating to instances based on their similarity to previously rated instances. Traditional classification and regression techniques are not well-suited for ranking tasks since the labels are structured with a total order relation between them. Therefore, specialized algorithms for ranking learning have been developed.

In this Project, we present an alternative approach to ranking learning that directly maintains a totally ordered set via projections. Our algorithm uses projections to associate each ranking with a distinct sub-interval of the real numbers and adapts the support of each sub-interval while learning. We describe a simple and efficient online algorithm that manipulates the direction in which we project the instances and the division into sub-intervals concurrently.

To evaluate the performance of our algorithm, we compare it with seven pre-existing algorithms on four different data sets, including synthetic data [5], Each-Movie data set for collaborative filtering[7], IMDB movie data set for collaborative filtering[2], and TMDB movie data set[6]. Our results show that the P-Ranking algorithm outperforms other algorithms on the first two data sets, while the multi-class perceptron algorithm outperforms others on the last two data sets. We also propose a neural network approach for ranking and discuss some natural language processing algorithms that we used in our project.

We provide a survey of existing literature in Section 2. Our proposal for the project is described in Section 3. We give details on experiments in Section 5. A description of future work is given in Section 7. We conclude with a short summary and pointers to forthcoming work in Section 8.

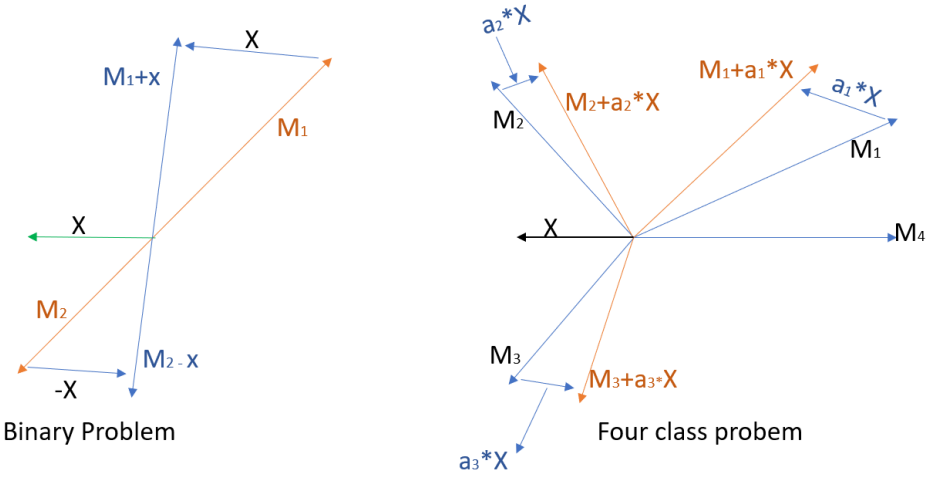
2 Literature Survey

2.1 From Binary to multi-class

The perceptron algorithm is an online algorithm for binary classification problems that have been widely used in machine learning[1]. The algorithm is based on maintaining a weight vector $w \in \mathbb{R}^n$ that is used to predict the label of an input instance x.let's assume we have only two label sets 1,2 now for given an input instance x, the perceptron algorithm predicts that its label is $\widehat{y} = 1$ if $w \cdot x \geq 0$ and $\widehat{y} = 2$ otherwise. The algorithm modifies the weight vector only when there is a prediction error, i.e., when the predicted label $\widehat{y}$ is not equal to the true label y. When there is a prediction error, the weight vector w is updated by adding x to w if the correct label is 1, and subtracting x from w if the correct label is 2.

To extend the perceptron algorithm to multi-class problems[4], a prototype matrix M is used, where each row of the matrix represents a vector for each class. For each input instance x, the algorithm calculates the similarity score between the instance and each of the k prototypes vector in M to predict the label. Specifically, the predicted label $\widehat{y}$ is the index of the row (prototype) of M that achieves the highest score, that is, $\widehat{y} = argmax_r M_r \cdot x$

If the predicted label $\widehat{y}$ is incorrect, i.e., $\widehat{y} \neq y$, the algorithm moves the row M_y toward x by replacing M_y with $M_y + x$, and moves away from x the rows $M_r (r \neq y)$ for which $M_r \cdot x \geq M_y \cdot x$. The indices of these rows constitute the error set E. The algorithms presented modify M such that the total change in units of x in the rows of M that are moved away from x is equal to the change of M_y (in units of x). Specifically, for the multi-class perceptron algorithm, M_y is replaced with $M_y + x$, and for each r in E, M_r is replaced with $M_r - \frac{1}{|E|}x$, where $|E|$ is the cardinality of the error set E



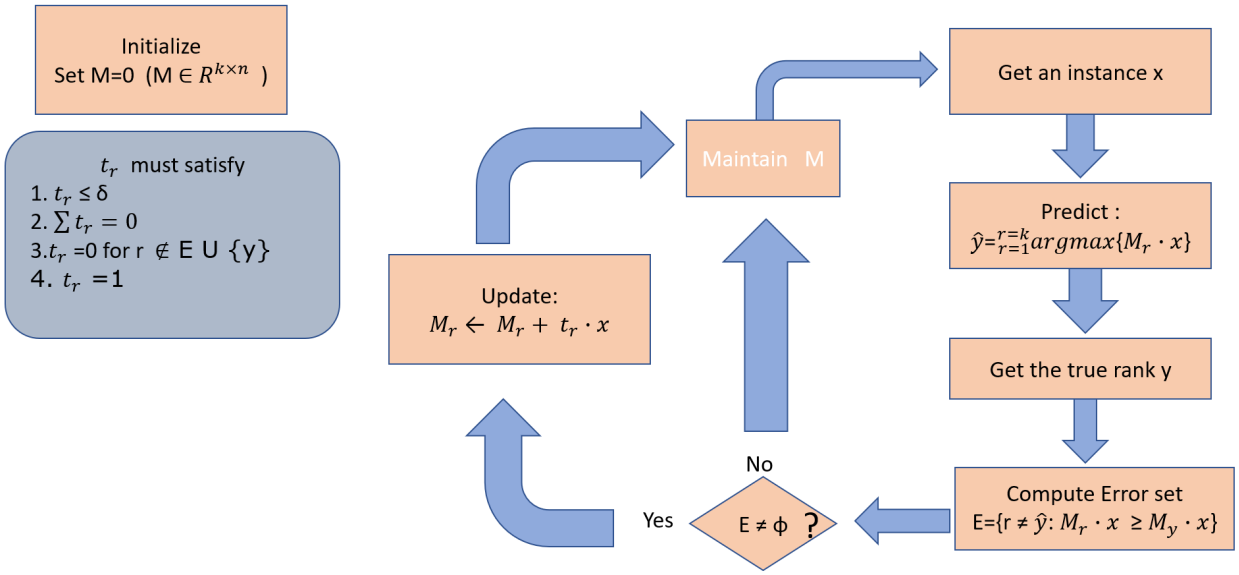
in the first figure predicted label of x is 2 but the actual label is 1 so first row of the protoptype matrix is updated by $M_1 + x$ and the second row is updated by $M_2 - x$. for more detail in the second picture describes how 4 class problem work in our setup lets a sample x is misclassified as 4 but their actual label is 1 so we update a first row of prototype matrix M by $M_1 + x$ and other two-row vector by $m_i + a_r \cdot x$

2.2 A Family of Additive Multi-class Algorithms

The family of multi-class algorithms[4] discuss before provides a way to update a matrix M that stores prototype vectors for each class. The algorithms are ultraconservative and update M only on rounds with prediction errors.To introduce more flexibility in the update, a vector of weights $\overline{\tau} = (\tau_1, ..., \tau_k)$ is introduced, and the update of the rth row is modified to $\overline{M}_r + \tau_r x$. The weights $\overline{\tau}$ are chosen such that the total change of the rows of M whose indices are from E is equal to the change in $\overline{M}_y$. The family of algorithms obtained by varying the values of $\overline{\tau}$ constitutes a whole family of multiclass algorithms.

Algorithm 1 Online Multiclass Perceptron Algorithm

Initialize: Set $M = 0$ ($M \in \mathbb{R}^{k \times n}$).
for $t = 1, 2, \ldots, T$ **do**
  Get a new instance $\overline{x}_t \in \mathbb{R}^n$.
  Predict $\hat{y}_t = \arg\max_{r=1}^{r=k} M_r \cdot \overline{x}_t$.
  Get a new label y_t.
  Set $E = \{r \mid r = y_t : M_r \cdot \overline{x}_t \geq M\overline{y_t} \cdot \overline{x}_t\}$.
  if $E \neq \emptyset$ **then**
    Update M by choosing any $\tau_1^t, \ldots, \tau_k^t$ that satisfy:
    1. $\tau_r^t \leq \delta r, y^t$ for $r = 1, \ldots, k$.
    2. $\sum_{r=1}^{k} \tau_r^t = 0$.
    3. $\tau_r^t = 0$ for $r \notin E \cup y^t$.
    4. $\tau_{y^t}^t = 1$.
  for $r = 1, 2, \ldots, k$ **do**
    Update: $M_r \leftarrow M_r + \tau_r^t \overline{x}^t$.
Output: $H(\overline{x}) = \arg\max_r M_r \cdot \overline{x}$.



Uniform Multiclass update

in the uniform multi-class update, we take τ_r as 1 if r is true label and update each row of matrix M which

belongs to Error set by equally weightage such that sum of τ_r is 0

$$\tau_r = \begin{cases} -\frac{1}{|E|} & \text{if } r \in E \\ 1 & \text{if } r = y \\ 0 & \text{otherwise} \end{cases}$$

Worst margin multiclass update

in this algorithm, we update as $M_r + x$ for the true label and update the row vector of M which corresponds to the predicted label by $M_r - x$ and we didn't update another row

$$\tau_r = \begin{cases} -1 & \text{if r=arg max}_s\{\overline{M}_s \cdot \overline{x}\ \} \\ 1 & \text{if } r = y \\ 0 & \text{otherwise} \end{cases}$$

similarity score multi-class update

The two instances mentioned above establish a constant value for τ_r, where r belongs to E, without considering the specific similarity scores obtained by each row. Alternatively, we can assign a value to τ for the promotion that reflects the surplus of the similarity score for each row in the error set compared to M_y. For example, we can determine the value of τ to be

$$\tau_r = \begin{cases} -\frac{[\overline{M}_r\overline{x} - \overline{M}_y \cdot \overline{x}]_+}{\sum_{i=1}^{k}[\overline{M}_r\overline{x} - \overline{M}_y \cdot \overline{x}]_+} & \text{if r}\neq y \\ 1 & \text{if } r = y \end{cases}$$

The Margin Infused Relaxed Algorithm (MIRA)

MIRA attempts to update a matrix M on each round based on the current input example x and its predicted label y. To do so, it chooses a vector $\bar{\tau}$ that minimizes the vector-norm of the new matrix M subject to the first two constraints only:

$$\forall r\ \tau_r \leq \delta_{r,y^t} : \text{The update is restricted to be within a certain margin of the correct label.} \quad (1)$$

$$\sum_r \tau_r = 1 : \text{The sum of the update vector must be 1.} \quad (2)$$

The optimization problem that MIRA solves on each round is a quadratic optimization problem, which can be expressed as:

$$\text{minimize } \ Q(\tau) = min_\tau \frac{1}{2}\sum_r ||\overline{M_r} + \tau_r \overline{x}^t||_2^2 \text{ subject to} \qquad \forall r\ \tau_r \leq \delta_{r,y} \text{ and } \sum_r \tau_r = 0 \quad (3)$$

that can be written as

$$\frac{1}{2}\sum_r ||\overline{M}_r + \tau_r\overline{x}||^2 = \frac{1}{2}\ \sum_r ||M_r||^2 + \sum_r \tau_r(\overline{M}_r \cdot \overline{x}) + \frac{1}{2}\sum_r \tau_r^2||\overline{x}||^2 \quad (4)$$

$$min_{\tau} Q(\tau) = \frac{1}{2} A \sum_{r=1}^{k} \tau_r^2 + \sum_{r=1}^{k} B_r \tau_r$$

subject to $\forall r \ \tau_r \leq \delta_{r,y}$ and $\sum_r \tau_r = 0$(5)

where $A = x^T x$ and $B_r = \overline{M}_r \cdot \overline{x}$.

One important property of the MIRA algorithm is that it is ultraconservative, meaning that it always chooses the most cautious or conservative update possible. This is because the algorithm automatically satisfies the third constraint of a family of algorithms from a previous section, which restricts the update to be within a certain margin of the correct label. If the input example is correctly classified, no update takes place and the vector $\bar{\tau}$ is the zero vector.

Overall, MIRA is a powerful algorithm that can be used for multiclass classification problems and is able to make updates to the matrix M based on the input example and its predicted label while still being ultraconservative.

after solving these optimization problem we got a close form of the solution as

$$\tau_r = min\{\theta^* - \frac{B_r}{A}, \delta_{y,r}\} \tag{6}$$

Initialize: Set $M = 0 \in \mathbb{R}^{k \times n}$.
Loop: For $t = 1, 2, \ldots, T$ do

- Get a new instance $\overline{x}^t$.
- Predict $\hat{y}t = \arg\max r \overline{M}_r \cdot \overline{x}^t$.
- Get a new label y_t.
- Find $\overline{\tau}_t$ that solves the following optimization problem:
  $\min_{\tau} \frac{1}{2} \sum_r ||\overline{M_r} + \tau_r \overline{x}^t||_2^2$
  subject to
  $\forall r \ \tau_r \leq \delta_{r,y}$
  and $\sum_r \tau_r = 0$
- Update: $Mr \leftarrow Mr + \bar{\tau}_r^t \overline{x}_t$ for $r = 1, 2, \ldots, k$.

Output: $H(\overline{x}) = \arg\max_r \overline{M}_r \cdot \overline{x}$, where $\overline{x}$ is the input to be classified.

Widrowhoff algorithm

The Widrow-Hoff algorithm[8], also known as the Least Mean Squares (LMS) algorithm, is a widely used algorithm in machine learning and signal processing. It is a type of online learning algorithm that is used for supervised learning tasks.

The goal of the Widrow-Hoff algorithm[8] is to find the optimal weights for a linear regression model. These weights are adjusted iteratively based on the error between the predicted output and the actual output. The algorithm uses the gradient descent technique to update the weights.

The Widrow-Hoff algorithm works by taking in an input vector $\boldsymbol{x}$ and a corresponding target output value y. The algorithm then makes a prediction based on the current weights and calculates the error between

the predicted output $\hat{y}$ and the actual output y. The error is then used to adjust the weights such that the error is minimized.

The weight update rule in the Widrow-Hoff algorithm is given by the following equation:

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \alpha(y - \hat{y})\boldsymbol{x} \tag{7}$$

where $\boldsymbol{w}(t)$ is the weight vector at iteration t, α is the learning rate which controls the step size of the weight update, y is the target output value, $\hat{y}$ is the predicted output value, and $\boldsymbol{x}$ is the input vector.

The algorithm continues to iterate over the training data until the weights converge to a stable value or until a stopping criterion is met.

However, the Widrow-Hoff algorithm has some limitations. It can be sensitive to the choice of the learning rate and may converge slowly or not at all if the learning rate is too high or too low. Additionally, the algorithm assumes that the input features are linearly related to the target output, which may not always be the case.

A few experiments on a data-set are provided which indicate the usefulness of the approach.

3 Methods and Approaches

all algorithms as discussed before is time-consuming and the number of the parameter is large for multi-class perceptron[4] it takes a 1xn vector for each class that makes it kxn matrix in the Widrowhoff algorithm[8] it takes only n parameter but doesn't perform well so we come up with a new algorithm called P-Ranking that have less parameter and it performs well compared to all algorithm discussed before.

3.1 Work done before mid-term project review

here we are discussing P-Rank algorithm.[3] Each instance is represented by a vector in $\mathbb{R}^n$, and is associated with a rank from a finite set Y with a total order relation. The order relation over Y induces a partial order over the instances, where one instance is preferred over another if its rank is higher. If two instances have the same rank, they are considered equivalent. The goal is to develop algorithms that can rank new instances as they arrive in an online setting, based on the partial order induced by the previous instances.

A ranking rule is a function that maps instances to rank values. The ranking rules discussed in this article use a vector w and a set of thresholds ($b_1 to b_{k-1}$) to assign ranks to new instances. To predict the rank of a new instance, the rule computes the inner product of the instance with the vector w, and then checks which threshold the result falls below. This assigns the instance to a rank, with all instances falling between adjacent thresholds being assigned the same rank. The minimum threshold for which the result falls below is the predicted rank for the instance. This approach divides the instance space into parallel, equally ranked regions

the author describes an algorithm that works in rounds, where on each round, the algorithm receives an instance and predicts a rank. The algorithm then updates its ranking rule based on the correct rank value. The goal of the algorithm is to minimize the ranking loss, which is the number of thresholds between the true rank and the predicted rank. The ranking loss after T rounds is the cumulative difference between the predicted and true rank values. The algorithm updates its ranking rule only when it makes a mistake, and it is called a conservative algorithm. The predicted rank value is compared to the true rank value, and

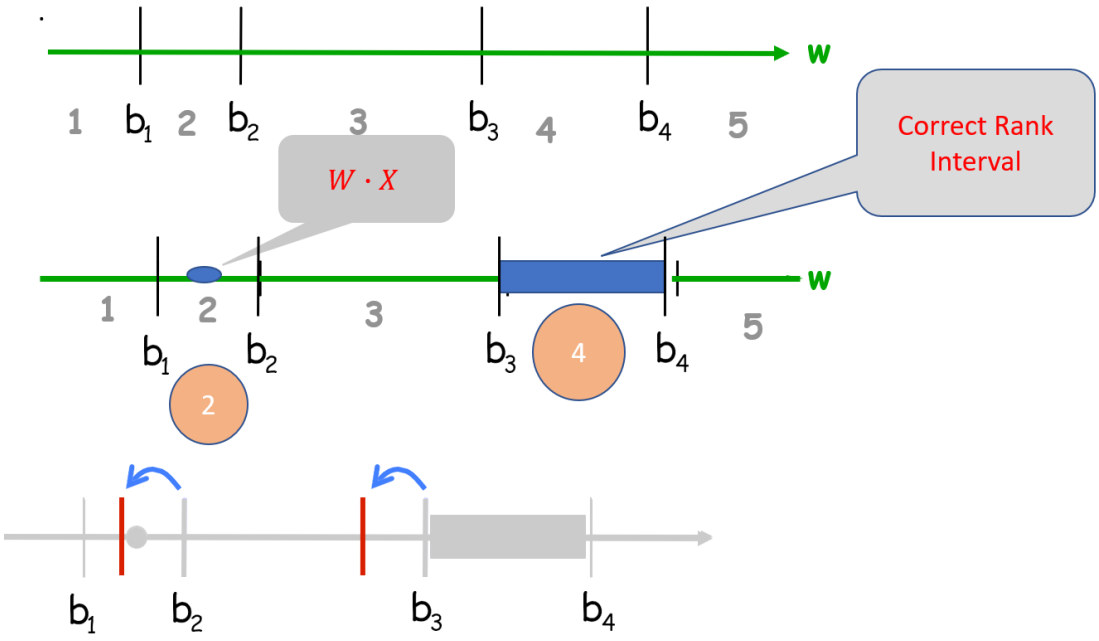
if they are not equal, the ranking rule is updated to improve the algorithm's performance. The ranking loss is calculated using the formula:

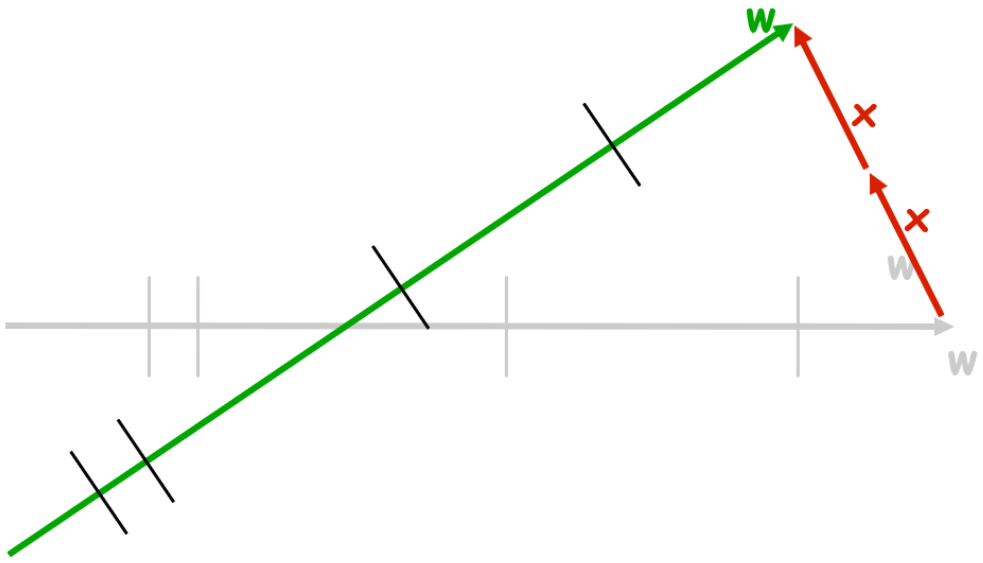
Ranking loss = $\sum_{t=1}^{T} |\hat{y}^t - y^t|$, where T is the total number of rounds, $\hat{y}_t$ is the predicted rank, and y_t is the true rank.

To predict a ranking, we expand the actual rank y into $k-1$ virtual variables y_r. We set $y_r = +1$ when $w\bar{x} > b_r$ and $y_r = -1$ when $w\bar{x} < b_r$, where b_r is the threshold for the r-th rank. The predicted rank is correct if $\forall r, y_r(w\bar{x} - b_r) > 0$.

If the algorithm misclassifies an instance $(\bar{x}, y)$, we need to adjust the thresholds and weights. Specifically, we modify the thresholds where the weight times the instance is on the wrong side of b_r, i.e., where $y_r(w \cdot \bar{x} - b_r) \leq 0$. We replace the incorrect thresholds with the difference between the threshold and the corresponding virtual variable, i.e., $b_r - y_r$. We also update the weight based on true label is how far from the actual label i.e., $w = w + r : y_r(w \cdot \bar{x} - b_r) \leq 0x$

An illustration of the update rule is given in Figure In the example, we used the set $Y = 1, ..., 5$. (Note that $b_5 = \infty$ is omitted from all the plots in Figure) The correct rank of the instance is $y = 4$, and thus the value of $\bar{w} \cdot \bar{x}$ should fall in the fourth interval, between b_3 and b_4. However, in the illustration, the value of $\bar{w} \cdot \bar{x}$ fell between b_1 and b_2 and the predicted rank is $\hat{y} = 2$. The threshold values b_2, and b_3 are a source of the error since the value of b_2, b_3 is higher than $\bar{w} \cdot \bar{x}$. To compensate for the mistake, the algorithm decreases b_2, and b_3 by a unit value and replaces them with $b_2 - 1$, and $b_3 - 1$. It also modifies $\bar{w}$ to be $\bar{w} + 2\bar{x}$ since $r : y_r(\bar{w} \cdot \bar{x} - b_r) \leq 0$ and $y_r = 2$. Thus, the inner product $\bar{w} \cdot \bar{x}$ increases by $2\bar{x}$.





Algorithm 2 The PRank Algorithm

Number of iterations T, rank-value $x_t \in \mathbb{R}^n$, label y_t, initialization $w_I = 0$, $b_1 = \cdots = b_{L_1} = 0$, $b_l = 0$
Prediction $H(x)$

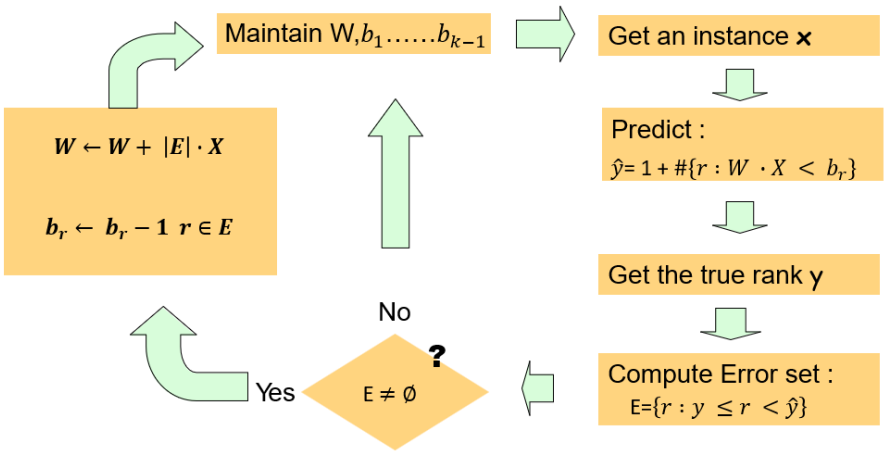
for $t = 1$ to T **do** $f_l \leftarrow \min_{r \in 1,\ldots,k} r : w_t \cdot x_t - b_l < \epsilon$; $y_t \leftarrow$ get new label;
  if $f_l \neq y_t$ **then**
    for $r = 1$ to $k - 1$ **do**
      if $y_t \leq r$ **then** $y_r \leftarrow -1$;
      else $y_r \leftarrow 1$;
      for $r = 1$ to $k - 1$ **do**
        if $(w_t \cdot x_t - b_l)y_r \leq 0$ **then** $T_r \leftarrow y_r$;
        else $T_r \leftarrow 0$; $\quad w_{t+1} \leftarrow w_t + \eta \sum_{r=1}^{k-1} T_r x_t$;
        for $l = 1$ to $k - 1$ **do** $b_{l+1} \leftarrow b_l - T_l$;
$H(x) \leftarrow \min_{r \in 1,\ldots,k} r : w_{T+1} \cdot x - b_{l+1} < \epsilon$;
return $H(x)$;

3.2 Work done after mid-term project review

after mid sem I implemented code on two data-set IMBD and TMBD movie review Data-set Both are text data-set so I have used natural language processing tool that i explained here

StopWord removed from data

Stopwords are words that are commonly used in natural language but are not significant for text analysis.



They include prepositions, pronouns, conjunctions, and articles. In natural language processing, stopwords removal is a crucial step that ensures the accuracy of the results of text analysis. It is done to focus on the essential words in a text that carry more meaning, such as keywords and entities. The removal of stopwords enables algorithms to avoid noise and, therefore, can improve the accuracy of NLP tasks such as sentiment analysis, text classification, and information extraction

Stopword removal is important because it simplifies the text analysis process. As mentioned earlier, removing these words reduces the noise and allows algorithms to focus on the significant words in a text. It also improves the speed of analysis, as it reduces the size of the data that needs to be analyzed. Stopwords are common and therefore occur frequently in text, so their removal significantly reduces the computational power required to process the text. Consequently, stopword removal helps to speed up the analysis process and makes it more efficient.

Porter Stemmer

The Porter Stemmer is a popular algorithm used in natural language processing to convert words to their base or root form. It was developed by Martin Porter in the 1980s and is widely used in various text analysis and information retrieval tasks.

The algorithm works by applying a series of transformation rules to a given word, with the goal of reducing the word to its base form. These rules involve removing suffixes and applying different patterns depending on the part of speech of the word.

For example, the word "playing" would be reduced to "play" using the Porter-Stemmer algorithm. The suffix "ing" is removed, and the stem "play" is returned.

The Porter-Stemmer algorithm is particularly useful for tasks such as search engine optimization, document classification, and sentiment analysis. It helps to reduce the dimensionality of text data by collapsing words with similar meanings into a single base form.

However, the algorithm has some limitations. It can sometimes over-stem words, reducing them to a base form that is too general and losing some of their meaning. Additionally, it is not always effective in handling irregular forms of words, such as "children" or "geese."

Count Vectorize

A count vectorizer is a commonly used technique in natural language processing (NLP) that converts text

data into numerical form. It is a process of converting a collection of text documents to a matrix of token counts. The count vectorizer creates a vocabulary of all unique words in the corpus and then counts the occurrences of each word in each document. The result is a sparse matrix where each row represents a document and each column represents a word in the vocabulary.

Count vectorizer is useful for text classification tasks like sentiment analysis, topic modeling, and spam filtering. It helps in converting text data into a numerical format that can be easily processed by machine learning algorithms. By using a count vectorizer, we can extract important features from text data and create a model that can predict the class or category of a new document. Give details on the work done after the mid-term project review. Explain how it is significant. Explain the ML method used and the changes you tried. Explain all other required details.

3.3 Neural Network Approach of Ranking Algorithm:

this approach for the ordinal regression dataset is different from the standard classification neural network in that it considers the order of categories. It classifies a data point into all lower-order categories as well, which is different from standard classification that predicts the probability of a data point belonging to a single category. This proposed approach uses a target vector that sets all elements up to category k to 1 and the rest to 0, where k is the ordinal category of the data point. The goal of the proposed approach is to learn a function to map input vector x to a probability vector where elements up to category k are close to 1 and the rest are close to 0. This encoding is related to cumulative probit models, which consider the proportion of categories that x belongs to, starting from category 1.

The neural network has d input nodes and K output nodes, where d is the number of dimensions of the input feature vector and K is the number of ordinal categories. The neural network can have one or more hidden layers, with transfer functions for the hidden nodes that can be a linear function, sigmoid function, or tanh function.

The output nodes use independent sigmoid functions to estimate the probability of the input belonging to each ordinal category. Specifically, the output node p_i uses the sigmoid function $\frac{1}{1+e^{-z_i}}$, where z_i is the net input to the output node. The target vector is a binary vector that sets the target value of output nodes corresponding to the input's true ordinal category. For example, if the input belongs to category k, then the target vector is (1, ..., 1, 0, ..., 0), where the first k elements are 1 and the rest are 0.

The neural network is trained using a cost function, which can be the relative entropy or square error between the predicted output vector and the target vector. The relative entropy cost function is given by $f_c = \sum_{i=1}^{k}(t_i log p_i) + (1 - t_i)log(1 - p_i)$, where t_i is the target value of the output node and p_i is the predicted probability output. The goal of training is to adjust the weights of the neural network to produce probability outputs that are as close as possible to the target vector, while also potentially imposing inequality constraints to ensure a monotonic relation between the output probabilities

The error function is $f_c = \frac{1}{2}\sum_{i=1}^{K}(t_i - p_i)^2$. Previous studies (Richard and Lippman, 1991) on neural network cost functions show that relative entropy and square error functions usually yield very similar results. In our experiments, we use square error function and standard back-propagation to train the neural network. The errors are propagated back to output nodes, and from output nodes to hidden nodes, and finally to input nodes.

Since the transfer function f_t of output node p_i is the independent sigmoid function $\frac{1}{1+e^{-z_i}}$, the derivative of f_t of output node O_i is $\frac{\partial f_t}{\partial z_i} = \frac{e^{-z_i}}{(1+e^{-z_i})^2} = \frac{1}{1+e^{-z_i}}\left(1 - \frac{1}{1+e^{-z_i}}\right) = p_i(1 - p_i)$. Thus, the net error propagated to output node p_i is $\frac{\partial f_c}{\partial p_i}\frac{\partial f_t}{\partial z_i} = t_i - p_i$ for relative entropy cost function, $\frac{\partial f_c}{\partial p_i}\frac{\partial f_t}{\partial z_i} = -2(t_i - p_i)p_i(1 - p_i)$ for square error cost function. The net errors are propagated through neural networks to adjust weights

using gradient descent as traditional neural networks do.

4 Data set Details

Synthetic Dataset

We generated 100 sets, each consisting of 7000 pairs of data points, denoted by (X_1, X_2), where both X_1 and X_2 are independently and uniformly generated from the unit square $[0, 1]$. To each data point, we assigned a ranking value y based on the following formula:

$y = max_r\{r : 10((x_1 - 0.5)(x_2 - 0.5)) + \xi > b_r\}$

where b is a list of five values $-\infty, -1, -0.1, 0.25, 1$, and ξ is a random noise value with a mean of 0 and a standard deviation of 0.125

Each Movie Data-set

The DEC Systems Research Center conducted an experiment with a collaborative filtering algorithm by running the Each-Movie recommendation service for 18 months. This service received ratings from 72,916 users for 1,628 movies, totaling 2,811,983 ratings. and all user identification removed

TMDB Movie Review Data-Set

The TMBD Data-set is a comprehensive collection of data related to 45,000 movies released on or before July 2017. The data-set includes information such as cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts, and vote averages.

The data set is divided into several files, with moviesmetadata.csv being the main file that contains information on all 45,000 movies. Additionally, there are files containing plot keywords, cast and crew information, and links to TMDB and IMDB IDs.

One of the most significant files in the data-set is ratings.csv, which contains 26 million ratings from 270,000 users for all 45,000 movies. These ratings are on a scale of 1-5 and were obtained from the official GroupLens website.It provides a comprehensive and diverse set of information on thousands of movies, including ratings from a large number of users, making it a powerful tool for movie recommendation systems, content analysis, and other data-driven approaches to movie research.

IMDB Movie Review Data-Set

The pre-processed Stanford IMDB movie review data-set in CSV format contains a collection of movie reviews with associated sentiment and rating values. The data-set has been pre-processed by converting the original folder structure into a CSV file using file-handling techniques. The original dataset, which is available on the Stanford AI Lab website, includes movie reviews from IMDB and their corresponding ratings. The ratings in the dataset range from 1 to 4 for negative reviews and from 7 to 10 for positive reviews.

The dataset also includes a sentiment column where a value of 1 indicates a positive sentiment and a value of 0 indicates a negative sentiment. Each record in the dataset contains a text review, a rating, and a sentiment value. The dataset can be used for sentiment analysis tasks, where the goal is to classify a given review as either positive or negative.

5 Experiments

for Synthetic Data

six different algorithms were used to predict the rank of each instance. The algorithm's predictions were converted into rank values by rounding the real-valued predictions to the closest rank value. The inner-product operation between each input instance and the hyperplanes maintained by the algorithms was computed using a non-homogeneous polynomial of degree 2, $K(X_1, X_2) = ((X_1, X_2) + 1)^2$.

At each time step, the ranking-loss for each algorithm was computed by comparing the predicted rank with the actual rank of the instance, and the accumulated loss was normalized by the instantaneous sequence length. The time-averaged loss after T rounds were computed using the formula $\frac{1}{T}\sum_{i=1}^{T}(y_t - \overline{y}_t)$, where y_t represents the true rank of the instance and $\overline{y}_t$ represents the predicted rank of the instance.

To increase the statistical significance of the results, the process was repeated 100 times, with a new random instance-rank sequence of length 7,000 being picked each time. The instantaneous losses across the 100 runs were averaged to obtain a more robust estimate of the ranking performance of each algorithm over the sequence. The computed losses were reported for $T = 1, ..., 7000$ to investigate the performance of the algorithms over time

For Each Movie Data-set

The goal of this study was to develop a prediction rule that can be used to learn correlations between the tastes of different viewers by predicting the "taste" of a random user based on their past ratings and the ratings of fellow viewers as features. The data set consisted of 7,542 viewers who had rated at least 100 movies out of a collection of 1623 titles, with each viewer rating each movie using a scale of 0, 0.2, 0.4, 0.6, 0.8, or 1. The ratings were linearly transformed by subtracting 0.5 from each rating to enable assigning a value of zero to movies that had not been rated.

The study was conducted using an online learning approach, where feature-rank pairs were fed one at a time. The experiment was repeated 500 times, with each iteration randomly selecting a target viewer for the rank. The study was repeated using viewers who had seen at least 200 movies (out of the same collection of 1623 titles)

For TMDB Movie Review Dataset

we preprocessed a dataset of TMDB movie reviews in order to make it more suitable for analysis. Specifically, we combined all relevant text data into a single column for each movie and calculated an average rating based on user ratings, rounded to one decimal place.

To clean the text data, we used packages from the Natural Language Toolkit (NLTK), including Stopword and Porter stemmer. This helped to remove noise from the data and focus on the most important information. Next, we used the Count Vectorizer package from Scikit-learn (SKLearn) to convert the cleaned text data into numerical data, which is more amenable to machine learning algorithms. and then implemented all algorithms as Discussed before for 200 Epoch

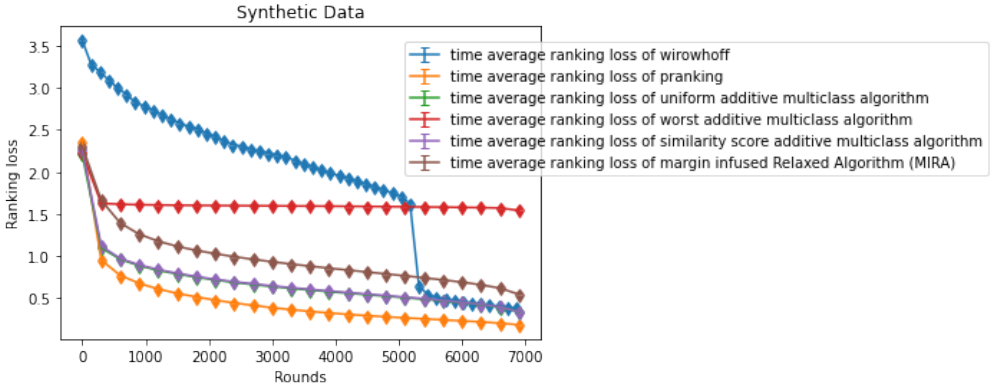
For IMDB Movie Review Data-set

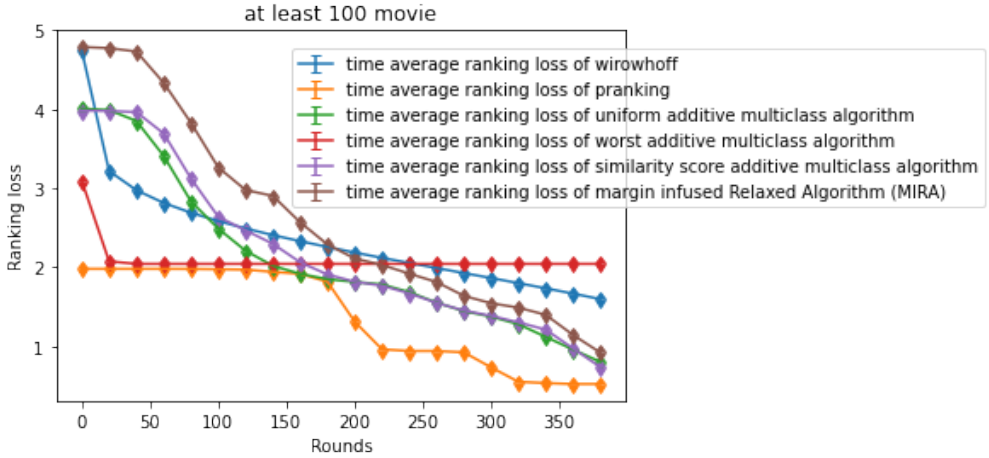
we examined IMDB movie reviews similar to the TMBD movie review dataset. We first preprocessed the data by combining relevant text data into a single column for each movie, and calculating an average rating based on user ratings, rounded to one decimal place. We also used packages from the Natural Language Toolkit (NLTK), including Stopword and Porter stemmer, to clean the text data and remove noise.

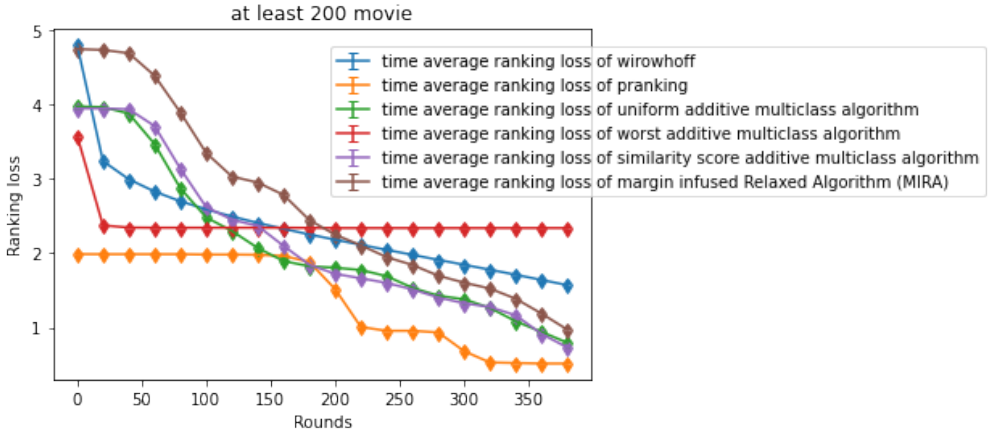
Next, we used the Count Vectorizer package from Scikit-learn (SKLearn) to convert the cleaned text data into numerical data. We then implemented eight different machine learning algorithms as Discussed Before

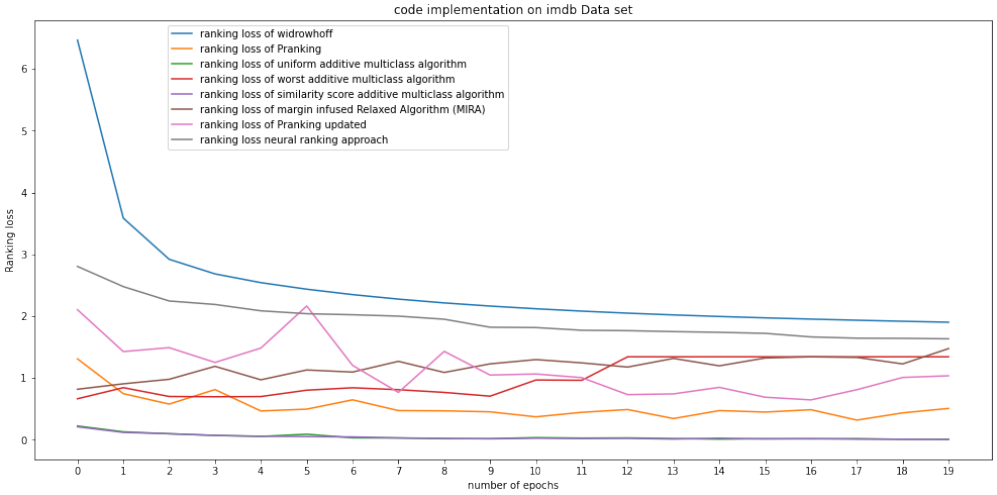
for 20 Epoch,

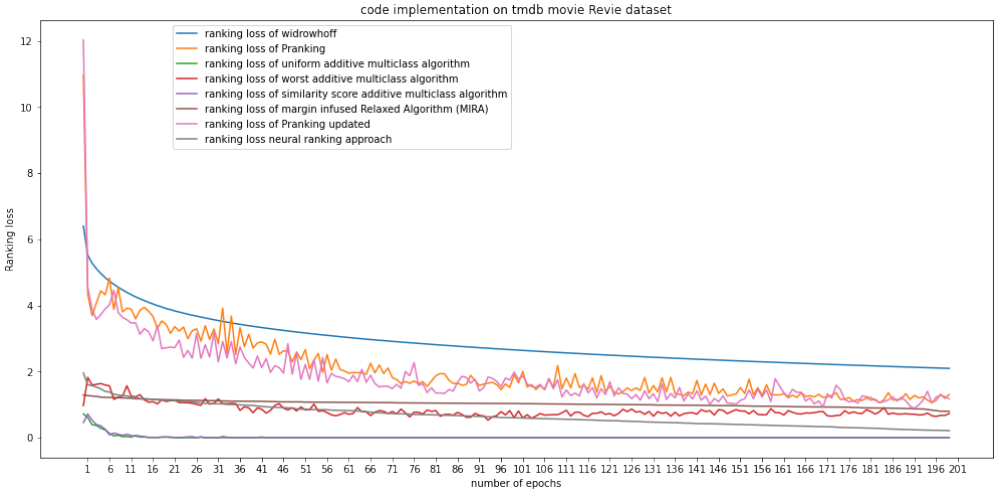
6 Results











we compared the performance of several machine learning algorithms on four different datasets: Synthetic, Each Movie, IMDB, and TMDB. Our results showed that for the first two datasets, Synthetic and Each Movie, the P-Ranking algorithm outperformed all other algorithms tested. This suggests that P-Ranking may be particularly effective for analyzing datasets with similar characteristics.

However, for the last two datasets, IMDB and TMDB, P-Ranking did not perform as well compared to the multi-class perceptron algorithm. This may be due to the specific features of these datasets, which may require different modeling techniques.

7 Future Work

there are several potential directions for further research in this area. One possibility is to explore the use of a pairwise approach for ranking, which may be useful in cases where we only care about the relative order of items and not their actual ranks. This approach could be particularly effective in situations where there are many items to be ranked, as it would reduce the computational complexity of the problem.

Another area for future work is the development of recommendation systems based on the insights gained from our analysis. By using machine learning algorithms to analyze movie reviews and ratings, we can gain insights into the factors that contribute to the success of a movie, and use this information to develop more effective recommendation systems. These systems could be used by filmmakers, critics, and audiences to identify movies that are likely to be successful or to identify movies that are similar to ones that a user has enjoyed in the past

8 Conclusion

In conclusion, our study compared the performance of several machine learning algorithms on four different datasets and found that the P-Ranking algorithm outperformed all others on the Synthetic and Each Movie datasets. However, for the IMDB and TMDB datasets, the Additive Multiclass algorithm performed well compared to P-Ranking.

Our results suggest that the effectiveness of different machine learning algorithms may vary depending on the specific characteristics of the dataset being analyzed. In particular, P-Ranking may be particularly effective when labels are parallelly separable in higher dimensions

References

[1] *18 The Perceptron: A Probabilistic Model for Information Storage and Organization (1958)*, pages 183–190. 2020.

[2] Nisarg Chodavadiya. Imbd movie review data. `https://www.kaggle.com/code/nisargchodavadiya/movie-review-analytics-sentiment-ratings/notebook`, 2021. Accessed on April 29, 2023.

[3] Koby Crammer and Yoram Singer. Pranking with ranking. In *NIPS*, 2001.

[4] Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3(Jan):951–991, 2003.

[5] Ralf Herbrich. Large margin rank boundaries for ordinal regression. *Advances in large margin classifiers*, pages 115–132, 2000.

[6] Kaggle. Tmdb movie review dataset dataset. `https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset`, 2017. Accessed on April 29, 2023.

[7] Paul McJones. Eachmovie Collaborative Filtering Dataset, DEC Systems Research Center,http://www.research.compaq.com/src/eachmovie/, 1997.

[8] Bernard Widrow and Marcian E. Hoff. Adaptive switching circuits. 1988.