Simple Trick to Train an Ordinal Regression with any Classifier

Muhammad May 15, 2019 · 4 min read ★

In this article I will show a simple method to tackle any ordinal regression (ordinal classification) problem using any existing binary classification algorithm, the method is developed by Eibe Frank and Mark Hal.

Original paper
(Eibe Frank and Mark Hal, lECML 2001. 12th European Conference)
https://www.cs.waikato.ac.nz/~eibe/pubs/ordinal_tech_report.pdf

Practical applications of machine learning sometimes involve a situation where the target values exhibit an order among different categories. However standard classification algorithms cannot make use of this ordering information, because they treat each class attribute as a set of unordered values [1].

This paper presents a simple trick that enables any standard classification algorithm to make use of ordering information in class attributes. The paper also shows that this simple trick outperforms naive classification approach, which treats each class as a set of unordered values.

Problem Definition

(Wikipedia) In statistics, **ordinal regression** (also called "**ordinal classification**") is a type of regression analysis used for predicting an ordinal variable, i.e. a variable whose value exists on an arbitrary scale where only the relative ordering between different values is significant.

Some examples of ordinal regression problems are predicting human preferences (strongly disagree to strongly agree), predict a temperature (Hot, Mild, Cold), predict a book/movie ratings (1 to 5).

Some Possible Approach

Some simple and naive way to tackle an ordinal regression problem is

1. Treat it as a regression problem
   If the ordinal value represent interval or ratio and we have that original interval/ratio value we can just treat it as a regression problem
   we fit a regression algorithms to the corresponding interval/ratio value that can be mapped to the actual ordinal value
   Cons: we can't use this approach if the ordinal value is not representing any continuous interval/ratio (like book/movie ratings), or if we don't have the original interval/ratio value.

2. Treat it as a standard classification problem
   We treat each ordinal value as an unordered set and fit a multiclass classification algorithm on it
   Cons: lose the ordering information of each class because the standard classification algorithm treat each class as a set of unordered values

Proposed Approach

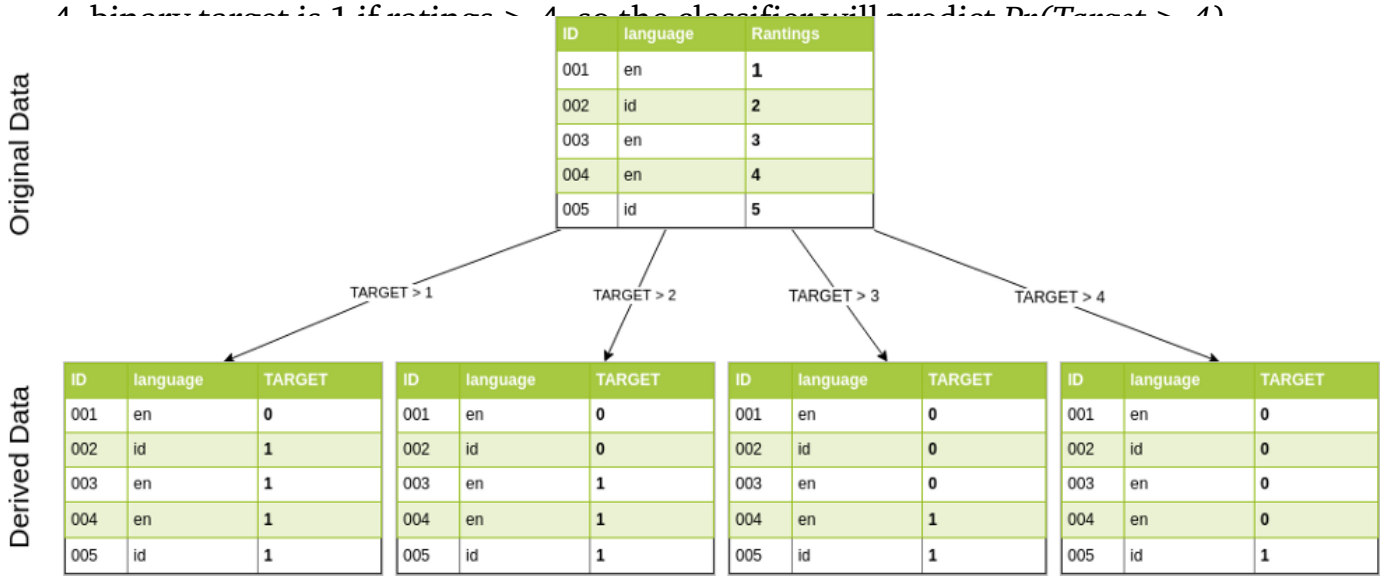
The proposed approach will work as long as we use a classifier that able to estimate output class probability

We can take advantage of the ordered class value by transforming a *k-class* ordinal regression problem to a *k-1* binary classification problem, we convert an ordinal attribute A* with ordinal value V1, V2, V3, … Vk into k-1 binary attributes, one for each of the original attribute's first $k - 1$ values. The ith binary attribute represents the test $A^* > Vi$ [1]

For example, if we have a prediction target with ordinal value movie ratings from 1

to 5 we can transform it to 4 binary classification problem such that

1. binary target is 1 if ratings > 1, so the classifier will predict *Pr(Target > 1)*
2. binary target is 1 if ratings > 2, so the classifier will predict *Pr(Target > 2)*
3. binary target is 1 if ratings > 4, so the classifier will predict *Pr(Target > 3)*
4. binary target is 1 if ratings > 4, so the classifier will predict *Pr(Target > 4)*

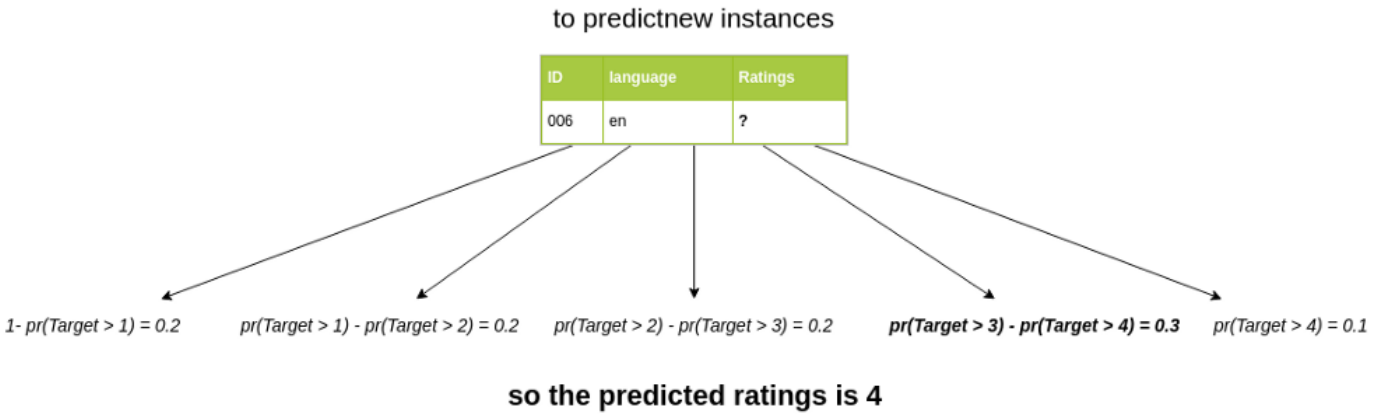


Example of the transformation to binary classification problem is done

After we trained 4 binary classifiers, we can use it to predict the probabilities of the ordinal value by using:

$Pr(y=1) = 1\text{-}Pr(Target > 1)$
$Pr(y=2) = Pr(Target>1)\text{-}P(Target > 2)$
$Pr(y=3) = Pr(Target>2)\text{-}P(Target > 3)$
$Pr(y=4) = Pr(Target>3)\text{-}P(Target > 4)$
$Pr(y=5) = Pr(Target>4)$

or generally, for the first ordinal value, we can use the first classifier to predict the probability of $V1 = 1$ — $Pr(y > V1)$
and we can use the last classifier to predict the probability of
$Vk = Pr(Target > Vk\text{-}1)$
and we can use in between classifier to predict the probability of
$Vi = Pr(Target > Vi) — Pr(Target > Vi\text{-}1)$



that way we are not losing its ordering information from the class label.

Python Implementation

We implement the trick described above by creating `OrdinalClassifier` class that will train k-1 binary classifier when `fit` is called, and will return predicted class if `predict` is called. During training (`fit`) phase `OrdinalClassifier` will store each of its trained binary classifiers to a python dictionary.

The parameter that defines our OrdinalClassifier is
clf: any sklearn classifier that implements `predict_proba` method in it

for the sake of example to create an OrdinalClassifier that use DecisionTree with max_depth = 3 would be:

```
clf = OrdinalClassifier(DecisionTreeClassifier(max_depth=3))
```

An example of Ordinal Classifier implemented in python:

```python
from sklearn.base import clone


class OrdinalClassifier():

    def __init__(self, clf):
        self.clf = clf
        self.clfs = {}

    def fit(self, X, y):
        self.unique_class = np.sort(np.unique(y))
        if self.unique_class.shape[0] > 2:
            for i in range(self.unique_class.shape[0]-1):
                # for each k - 1 ordinal value we fit a binary classification proble
                binary_y = (y > self.unique_class[i]).astype(np.uint8)
                clf = clone(self.clf)
                clf.fit(X, binary_y)
                self.clfs[i] = clf

    def predict_proba(self, X):
        clfs_predict = {k:self.clfs[k].predict_proba(X) for k in self.clfs}
        predicted = []
        for i,y in enumerate(self.unique_class):
            if i == 0:
                # V1 = 1 - Pr(y > V1)
                predicted.append(1 - clfs_predict[y][:,1])
            elif y in clfs_predict:
                # Vi = Pr(y > Vi-1) - Pr(y > Vi)
                 predicted.append(clfs_predict[y-1][:,1] - clfs_predict[y][:,1])
            else:
                # Vk = Pr(y > Vk-1)
                predicted.append(clfs_predict[y-1][:,1])
        return np.vstack(predicted).T

    def predict(self, X):
        return np.argmax(self.predict_proba(X), axis=1)
```

1. **__init__:** constructor, simply define our `OrdinalClassifier` and initiate `clfs` as a dictionary that will be used to store our k-1 binary classifier

2. **fit**: first we store each unique label available, then for first k-1 value, for each iteration, we transform its label to a binary label that represents the test A* > Vi `binary_y = (y > self.unique_class[i].astype(np.uint8))` , then we fit a new

classifier on the transformed label (`binary_y`) finally, store our classifier to the `clfs` dictionary with `i` as it's key

3. **predict_proba**: to get predicted probability of each class: first we get all prediction probability from all of our classifiers that stored on `clfs` after that simply enumerate all possible class label and append its prediction to our `predicted` list, after that, return it as a numpy array

4. **predict**: to get the predicted class simply take the `argmax` of our `predict_proba`

Thanks to Sabber Ahamed.