AP4: Modeling and Performance

Predictive Modeling

Sandra Zavala / **INFO 159** Published at Apr 22, 2022 **Private**

Ordinary Regression

Ordinal regression is a classification method for categories on an ordinal scale -- e.g. [1, 2, 3, 4, 5] or [G, PG, PG-13, R]. This notebook implements ordinal regression using the method of Frank and Hal 2001, which transforms a k-multiclass classifier into k-1 binary classifiers (each of which predicts whether a data point is above a threshold in the ordinal scale -- e.g., whether a movie is "higher" than PG). This method can be used with any binary classification method that outputs probabilities; here L2-regularizaed binary logistic regression is used.This notebook trains a model (on train.txt), optimizes L2 regularization strength on dev.txt, and evaluates performance on test.txt. Reports test accuracy with 95% confidence intervals.

```
from scipy import sparse
from sklearn import linear_model
from collections import Counter
import numpy as np
import operator
import nltk
import math
from scipy.stats import norm
from sklearn.feature_extraction.text import CountVectorizer
import pandas as pd
import nltk.stem
```

```
!python -m nltk.downloader punkt
```

```
/usr/local/lib/python3.7/runpy.py:125: RuntimeWarning: 'nltk.downloader' found in sys.modules after import of package 'nltk', but prior to execution of 'nltk.downloade
  warn(RuntimeWarning(msg))
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
def load_ordinal_data(filename, ordering):
    X = []
    Y = []
    orig_Y=[]
    for ordinal in ordering:
        Y.append([])

    with open(filename, encoding="utf-8") as file:
        for line in file:
            cols = line.split("\t")
            idd = cols[0]
            label = cols[1].lstrip().rstrip()
            text = cols[2]

            X.append(text)

            index=ordering.index(label)
            for i in range(len(ordering)):
                if index > i:
                    Y[i].append(1)
                else:
                    Y[i].append(0)
            orig_Y.append(label)

    return X, Y, orig_Y
```

```
class OrdinalClassifier:

    def __init__(self, ordinal_values, feature_method, trainX, trainY, devX, devY, testX, testY, orig_trainY, orig_devY, orig_testY):
        self.ordinal_values=ordinal_values
        self.feature_vocab = {}
        self.feature_method = feature_method
        self.min_feature_count=2
        self.log_regs = [None]* (len(self.ordinal_values)-1)

        self.trainY=trainY
        self.devY=devY
        self.testY=testY
```

```
        self.orig_trainY=orig_trainY
        self.orig_devY=orig_devY
        self.orig_testY=orig_testY

        self.trainX = self.process(trainX, training=True)
        self.devX = self.process(devX, training=False)
        self.testX = self.process(testX, training=False)

    # Featurize entire dataset
    def featurize(self, data):
        featurized_data = []
        for text in data:
            feats = self.feature_method(text)
            featurized_data.append(feats)
        return featurized_data

    # Read dataset and returned featurized representation as sparse matrix + label array
    def process(self, X_data, training = False):

        data = self.featurize(X_data)

        if training:
            fid = 0
            feature_doc_count = Counter()
            for feats in data:
                for feat in feats:
                    feature_doc_count[feat]+= 1

            for feat in feature_doc_count:
                if feature_doc_count[feat] >= self.min_feature_count:
                    self.feature_vocab[feat] = fid
                    fid += 1

        F = len(self.feature_vocab)
        D = len(data)
        X = sparse.dok_matrix((D, F))
        for idx, feats in enumerate(data):
            for feat in feats:
                if feat in self.feature_vocab:
                    X[idx, self.feature_vocab[feat]] = feats[feat]

        return X


    def train(self):
        (D,F) = self.trainX.shape

        for idx, ordinal_value in enumerate(self.ordinal_values[:-1]):
            best_dev_accuracy=0
            best_model=None
            for C in [0.1, 1, 10, 100]:

                log_reg = linear_model.LogisticRegression(C = C, max_iter=1000)
                log_reg.fit(self.trainX, self.trainY[idx])
                development_accuracy = log_reg.score(self.devX, self.devY[idx])
                if development_accuracy > best_dev_accuracy:
                    best_dev_accuracy=development_accuracy
                    best_model=log_reg


            self.log_regs[idx]=best_model

    def test(self):
        cor=tot=0
        counts=Counter()
        preds=[None]*(len(self.ordinal_values)-1)
        for idx, ordinal_value in enumerate(self.ordinal_values[:-1]):
            preds[idx]=self.log_regs[idx].predict_proba(self.testX)[:,1]

        preds=np.array(preds)

        for data_point in range(len(preds[0])):


            ordinal_preds=np.zeros(len(self.ordinal_values))
            for ordinal in range(len(self.ordinal_values)-1):
                if ordinal == 0:
                    ordinal_preds[ordinal]=1-preds[ordinal][data_point]
                else:
                    ordinal_preds[ordinal]=preds[ordinal-1][data_point]-preds[ordinal][data_point]

            ordinal_preds[len(self.ordinal_values)-1]=preds[len(preds)-1][data_point]

            prediction=np.argmax(ordinal_preds)

            print(data_point, "actual: ",self.ordinal_values.index(self.orig_testY[data_point]) ,"prediction: ", prediction)
            counts[prediction]+=1
            if prediction == self.ordinal_values.index(self.orig_testY[data_point]):
                cor+=1
                print("correct")
```

```
            tot+=1

        return cor/tot
```

```
positive_words = open("positive_sentiment_dictionary.txt", encoding = "utf-8").read()
```

```
# #scratch work
with open('60/train.txt') as f:
    file_lines = f.readlines()

words_df = pd.DataFrame([string.split('\t') for string in file_lines])
words_df = words_df.rename(columns = {0:"id", 1:"annon", 2:"article"})
words_df
```

	id object	annon object	article object
	237 ▬▬▬▬▬▬ 0.6% 223 ▬▬▬▬▬▬ 0.6% 178 others ▬ 98.9%	3 ▬▬▬▬▬▬▬▬ 35% 4 ▬▬▬▬▬▬ 27.8% 2 others ▬▬ 37.2%	[Image: Anti-Sena BOSTON (CB… 1.1% 174 others ▬ 97.8%
170	34	4	Sandra Delgadillo o…
171	251	3	"Luis Tapia, a junior at…
172	99	3	Protesters in cities acros…
173	285	4	"Parents, sons and…
174	156	3	RANDY FURST — STAR TRIBUNE…
175	186	2	MIAMI (AP) - Hundreds of…
176	169	4	A dangerous gang culture…
177	300	3	A group of nearly a…
178	282	3	Groups protest…
179	91	1	LANSING — Shirell…

```
words_df
```

	id object	annon object	article object	article_clean	tokens object	tokens_clean…
	237 ▬▬▬▬▬▬ 0.6% 223 ▬▬▬▬▬▬ 0.6% 178 others ▬ 98.9%	3 ▬▬▬▬▬▬▬▬ 35% 4 ▬▬▬▬▬▬ 27.8% 2 others ▬▬ 37.2%	dozens of … 1.1% image antisenate 171 others ▬ 97.8%	dozens of … 1.1% image antisenate 171 others ▬ 97.8%	['dozens',… 1.1% ['image', … 1.1% 170 others ▬ 97.8%	['dozens',… 1.1% ['image', … 1.1% 170 others ▬ 97.8%
10	266	2	a crowd of about 50…	a crowd of about chante…	['crowd', 'chanted',…	['crowd', 'chanted',…
11	279	2	image antisenate…	image antisenate…	['image', 'antisenate'…	['image', 'antisenate'…
12	283	2	undocumented immigrant…	undocumented immigrant…	['undocumented',…	['undocumented',…
13	174	1	may day protests…	may day protests…	['may', 'day',…	['may', 'day',…
14	160	2	new orleans hundreds of…	new orleans hundreds of…	['new', 'orleans',…	['new', 'orleans',…
15	265	2	san diegans protest gop…	san diegans protest gop…	['san', 'diegans',…	['san', 'diegans',…
16	228	1	dozens of people…	dozens of people…	['dozens', 'people',…	['dozens', 'people',…
17	145	1	hundreds of workers…	hundreds of workers…	['hundreds', 'workers',…	['hundreds', 'workers',…
18	253	2	1 of 12 image 070118newsim…	of image families…	['image', 'families',…	['image', 'families',…
19	115	1	issaquah wash hundreds of…	issaquah wash hundreds of…	['issaquah', 'wash',…	['issaquah', 'wash',…

The goal is to create a counter to get the frequency of each word in the text. After we want to compare it to see how often positive words appear in the most frequent words(15 words). So in order to do that, the most common words are stop words(the, and) are removed as well as all punctuation except quotation marks because direct quotes adds to/evokes compassion to the reader. After, we remove words associated with images(jpeg, png) and digits because those would skew our model and we want to focus on words that contribute to our compassion metric. After getting the frequency, we look at the most 15 common words in the articles. We then iterate through words in given article and if word is present in frequent word and positive words dictionary, we append to positive words list. After, we look at the proportion of positive words that occur most frequently divided by total words in article to see if the most frequent tend to be positive. If they are greater than a our threshold - which in our case would be the proportion of positive to negative words -, they are assigned a score. We apply this exact method with a negative words dictionary.

We have also considered the edge cases of words that portray immigrants in a positive or negative light. If the word is "undocumented", "dreamer", "families", which humanizes immigrants, then we assign a score of . If the words convey a negative portrayal of immigrants such as "illegal","alien". In addition, we applied a bigram model to words that once again portrayed immigrants with positively or negatively.

```
# opens the negative words text
negative_words = open("negative-words.txt", encoding = "utf-8").read()
```

```
#self-compassion
#direct quotes -> syntax
#conjunction of words
#import a pos dictionary and compare to the data and if words
# from pos dictionary appear more than half of the text then its
#more likely to be rated 3 + i.e. if there is a cetrian amt of pos scores
from re import search
# Module to remove punctuation from string library
from string import punctuation

# Module to count word frequencies
from collections import Counter

# Module to help us remove stopwords
import nltk
nltk.download("stopwords")
nltk.download("averaged_perceptron_tagger")
from nltk.corpus import stopwords

def binary_bow_featurize(text):
    feats = {}
    #tokenize text
    tokens = nltk.word_tokenize(text)

    words = []
    #remove stopwords
    for word in tokens:
        word = word.lower()
        if word not in stopwords.words('english'):
            words.append(word)

    #remove punctuation except quotes and remove images
    for word in words:
        for char in punctuation:
            if word != '"' and word != '!' and char == word:
                words.remove(word)
        jpg = "\S+jpg"
        png = "\S+png"
        if search(jpg,word):
            words.remove(word)
        if search(png,word):
            words.remove(word)

    #remove digits
    for word in words:
        if word.isdigit():
            words.remove(word)

    #get positive words in text
    pos_words = []
    for word in words:
        if word in positive_words:
            pos_words.append(word)

    #get negative words in text
    neg_words = []
    for word in words:
        if word in negative_words:
            neg_words.append(word)

    #get proportion of positive to negative words in text
    prop_pos_to_neg = len(pos_words)/len(neg_words)

    #create Counter to count frequency of each word in text
    freq = Counter(words)
    total_words = len(freq)

    #find the 15 most common words in the text
    top_words = freq.most_common(15)
    top_features = [top_words[i][0] for i in range(len(top_words))] #15 most common words
    top_freq = [top_words[i][1] for i in range(len(top_words))] #15 most common words' frequency

    #get positive words amongst the top 15 most common words in the text
    pos_top_words = []
    for word in words:
       if word in top_features and word in positive_words:
           pos_top_words.append(word)

    # if the proportion of positive words amongst the 15 most common is greater than a certain threshold
    #assign the feature to a certain score
    if len(pos_top_words)/len(top_words) > 0.8:
        feats[word + " score2"] = 4
    if len(pos_top_words)/len(top_words) > 0.5:
        feats[word + " score2"] = 3
    if len(pos_top_words)/len(top_words) < 0.4:
```

```
        feats[word + " score2"] = 2
    if len(pos_top_words)/len(top_words) < 0.3:
        feats[word + " score2"] = 1

    #this section didn't work so we can remove it
    # if the proportion of positive to negative words is above/below a certain threshold, assign the feature
    #to a certain score
    # if prop_pos_to_neg > 0.75:
    #     feats[word + " score1"] = 4
    # if prop_pos_to_neg > 0.6:
    #     feats[word + " score1"] = 3
    # if prop_pos_to_neg < 0.3:
    #     feats[word + " score1"] = 2
    # if prop_pos_to_neg < 0.2:
    #     feats[word + " score1"] = 1

    # #if these certain terms appear in the 15 most common words and
    # #if the proportion of positive to negative words is above/below a certain threshold
    # #assign the feature to a certain score
    for word in words:
        if word in top_features and word == "dreamer" or word == "families" or word == "sanctuary" or word == "undocumented" or "trump" and prop_pos_to_neg >
            feats[word + " score3"] = 4
        if word in top_features and word == "dreamer" or word == "families" or word == "sanctuary" or word == "undocumented" or "trump" and prop_pos_to_neg >
            feats[word + " score3"] = 3
        if word == "illegal" or word == "alien" or word == "dangerous" or word == "criminal" or "trump" and prop_pos_to_neg < 0.3:
            feats[word + " score3"] = 2
        if word == "illegal" or word == "alien" or word == "dangerous" or word == "criminal" or "trump" and prop_pos_to_neg < 0.2:
            feats[word + " score3"] = 1

    #check length of text by checking frequency of words
    #Normally, shorter articles have less context and thus offer less room for compassion
    if len(words) < 15: #if article is too short to even find top 15 used words
        feats[word + "score4"] = 1
    if np.sum(top_freq)/len(tokens) > 0.8:
        feats[word + "score4"] = 1
    if np.sum(top_freq)/len(tokens) > 0.6:
        feats[word + "score4"] = 2
    if np.sum(top_freq)/len(tokens) > 0.5:
        feats[word + "score4"] = 3
    if np.sum(top_freq)/len(tokens) > 0.3:
        feats[word + "score4"] = 4

    # #this section doesn't improve accuracy so we can remove it
    # annon_4 = ["families", "us", "one", "march", "together", "parents", "community", "rights", "undocumented", "family", "separated", "support" ]
    # annon_3 = ["us", "march", "families", "daca", "program", "support", "united", "undocumented", "childhood", ""]
    # annon_2 = ["protestors", "antisenate", "law", "texas", "administration", "policy", "ice", "enforcement", "police", "new"]
    # annon_1 = ["houston", "bill", "hall", "supporters", "antisenate", "texas", "new", "policy", "ice", "enforcement", "boys"]
    # if word in top_features and word in annon_4:
    #     feats[word + "score3"] = 4
    # if word in top_features and word in annon_3:
    #     feats[word + "score3"] = 3
    # if word in top_features and word in annon_2:
    #     feats[word + "score3"] = 2
    # if word in top_features and word in annon_1:
    #     feats[word + "score3"] = 1

    #getting pairs of words
    bigram_vectorizer = CountVectorizer(ngram_range = (0,2),
                                    token_pattern = r'\b\w+\b'
                                    )
    clean_text = text.lower()
    for word in clean_text:
        for char in punctuation:
            if word != '"' and word != '!' and char == word:
                clean_text.replace(word, " ")
        if word.isdigit():
            clean_text.replace(word, " ")

    bigramv = bigram_vectorizer.fit_transform([clean_text]).toarray()
    text_features = bigram_vectorizer.get_feature_names()
    neg_feats = ["illegal immigrants", "illegal aliens", "theyre dangerous", "abuse welfare", "proud boys", "support police", "criminal aliens", "gang membe
    pos_feats = ["bully trump", "citizenship rights", "womens rights", "children separated","families together", "terrorize community", "peaceful protest", "
    for word in text_features:
        if word in pos_feats and prop_pos_to_neg > 0.75:
            feats[word + "score6"] = 4
        if word in pos_feats and  prop_pos_to_neg > 0.6:
            feats[word + "score6"] = 3
        if word in neg_feats and prop_pos_to_neg < 0.4:
            feats[word + "score6"] = 2
        if word in neg_feats and prop_pos_to_neg < 0.3:
            feats[word + "score6"] = 1


    return feats
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
```

```
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

```
common_words= ["immigrants", "immigration", "immigrant", "trump", "rally", "border", "police", "rally", "children", "day", "protest", "people", "city", "prot
#finding what words are frequent for which annotations
txt = []
words_4 = words_df[words_df["annon"] == "4"].reset_index(drop = "True")
for j in range(len(words_4)):
    for k in range(len(words_4["tokens_cleaned"][j])):
        if words_4["tokens_cleaned"][j][k] not in common_words:
            txt.append(words_4["tokens_cleaned"][j][k])

annon_4 = Counter(txt)
annon_4.most_common(50)
annon_4 = ["families", "us", "one", "march", "together", "parents", "community", "rights", "undocumented", "family", "separated", "support" ]
```

```
KeyError: 'tokens_cleaned'
```

Show error details

```
txt = []
words_3 = words_df[words_df["annon"] == "3"].reset_index(drop = "True")
for j in range(len(words_3)):
    for k in range(len(words_3["tokens_cleaned"][j])):
        if words_3["tokens_cleaned"][j][k] not in common_words:
            txt.append(words_3["tokens_cleaned"][j][k])

annon_3 = Counter(txt)
annon_3.most_common(50)
annon_3 = ["us", "march", "families", "daca", "program", "support", "united", "undocumented", "childhood", ""]
```

```
KeyError: 'tokens_cleaned'
```

Show error details

```
txt = []
words_2 = words_df[words_df["annon"] == "2"].reset_index(drop = "True")
for j in range(len(words_2)):
    for k in range(len(words_2["tokens_cleaned"][j])):
        if words_2["tokens_cleaned"][j][k] not in common_words:
            txt.append(words_2["tokens_cleaned"][j][k])

annon_2 = Counter(txt)
annon_2.most_common(50)
annon_2 = ["protestors", "antisenate", "law", "texas", "administration", "policy", "ice", "enforcement", "police", "new"]
```

```
txt = []
words_1 = words_df[words_df["annon"] == "1"].reset_index(drop = "True")
for j in range(len(words_1)):
    for k in range(len(words_1["tokens_cleaned"][j])):
        if words_1["tokens_cleaned"][j][k] not in common_words:
            txt.append(words_1["tokens_cleaned"][j][k])

annon_1 = Counter(txt)
annon_1.most_common(50)
annon_1 = ["houston", "bill", "hall", "supporters", "antisenate", "texas", "new", "policy", "ice", "enforcement", "boys"]
```

```
def confidence_intervals(accuracy, n, significance_level):
    critical_value=(1-significance_level)/2
    z_alpha=-1*norm.ppf(critical_value)
    se=math.sqrt((accuracy*(1-accuracy))/n)
    return accuracy-(se*z_alpha), accuracy+(se*z_alpha)
```

```
def run(trainingFile, devFile, testFile):
    trainX, trainY=load_data(trainingFile)
    devX, devY=load_data(devFile)
    testX, testY=load_data(testFile)

    simple_classifier = Classifier(binary_bow_featurize, trainX, trainY, devX, devY, testX, testY)
    simple_classifier.train()
    accuracy=simple_classifier.test()


    lower, upper=confidence_intervals(accuracy, len(devY), .95)
    print("Test accuracy for best dev model: %.3f, 95%% CIs: [%.3f %.3f]\n" % (accuracy, lower, upper))

    simple_classifier.printWeights()
```

```
def run(trainingFile, devFile, testFile, ordinal_values):


    trainX, trainY, orig_trainY=load_ordinal_data(trainingFile, ordinal_values)
    devX, devY, orig_devY=load_ordinal_data(devFile, ordinal_values)
    testX, testY, orig_testY=load_ordinal_data(testFile, ordinal_values)

    simple_classifier = OrdinalClassifier(ordinal_values, binary_bow_featurize, trainX, trainY, devX, devY, testX, testY, orig_trainY, orig_devY, orig_testY
    simple_classifier.train()
    accuracy=simple_classifier.test()

    lower, upper=confidence_intervals(accuracy, len(devY[0]), .95)
    print("Test accuracy for best dev model: %.3f, 95%% CIs: [%.3f %.3f]\n" % (accuracy, lower, upper))
```

```
# gid=2
# trainingFile = "splits/%s/train.txt" % gid
# devFile = "splits/%s/dev.txt" % gid
# testFile = "splits/%s/test.txt" % gid

trainingFile = "60/train.txt"
devFile = "60/dev.txt"
testFile = "60/test.txt"

# ordinal values must be in order *as strings* from smallest to largest, e.g.:
# ordinal_values=["G", "PG", "PG-13", "R"]

ordinal_values=["1", "2", "3", "4"]

run(trainingFile, devFile, testFile, ordinal_values)
```

```
39 actual:  2 prediction:  2
correct
40 actual:  1 prediction:  2
41 actual:  2 prediction:  2
correct
42 actual:  2 prediction:  1
43 actual:  1 prediction:  2
44 actual:  0 prediction:  2
45 actual:  2 prediction:  1
46 actual:  3 prediction:  2
47 actual:  2 prediction:  2
correct
48 actual:  1 prediction:  1
correct
49 actual:  3 prediction:  1
50 actual:  0 prediction:  2
51 actual:  1 prediction:  2
52 actual:  1 prediction:  2
53 actual:  1 prediction:  3
54 actual:  1 prediction:  1
correct
55 actual:  3 prediction:  2
56 actual:  0 prediction:  2
57 actual:  0 prediction:  0
correct
58 actual:  2 prediction:  2
correct
59 actual:  1 prediction:  0
Test accuracy for best dev model: 0.383, 95% CIs: [0.260 0.506]
```

Citation for positive sentiment dictionary used in this notebook: Minqing Hu and Bing Liu. "Mining and Summarizing Customer Reviews." ; Proceedings of the ACM SIGKDD International Conference on Knowledge ; Discovery and Data Mining (KDD-2004), Aug 22-25, 2004, Seattle, ; Washington, USA,

Citation for negative sentiment dictionary used in this notebook: Minqing Hu and Bing Liu. "Mining and Summarizing Customer Reviews." ; Proceedings of the ACM SIGKDD International Conference on Knowledge ; Discovery and Data Mining (KDD-2004), Aug 22-25, 2004, Seattle, ; Washington, USA,

Logistic Regression

Analysis

Question we respond - we need to only narrow down to 1/2 i think

Does your model learn features of the phenomenon that you didn't consider in your guidelines that might cause you to rethink the category boundaries? (See Long and So 2016 (Links to an external site.) for an example.)

Our model learned features that we did not consider in our guidelines. For example, the frequency of positive words within an article can help determine whether the article shows sympathy for immigrants. Where as we focus more on forms of syntax such as the use of direct quotes to have readers feel compassion for immigrants and women as well as the use of anecdotal stories. Adding this additional feature of positive words associated with compassion can improve the model's performance. We also included a dictionary of negative words in order to balance the data that our model evaluates by accounting for the articles that have a negative and less sympathetic portrayal of immigrants.

We have also come to the conclusion that measuring compassion is a complex metric to model. The way in which our articles were annotated did not fully account for this complexity which is why it was difficult to measure our categorical boundaries. Our models prediction were often + or - 1 from the actual score. For example, one way that compassion is measured is through the use of words to describe immigrants. Even when using a bigram model by pairing words that most evoke compassion or negatively portray immigrants, the low accuracy reflected demonstrates how compassion can be difficult to measure and predict. A large part of the low accuracy of our classifier comes down to flaws with our annotation guidelines. Different people from different background have different levels of what evokes feelings of being moved from certain text. Additionally, biases the reader has towards certain viewpoints can result in different annotators labelling the same document on different levels of compassion simply because they can relate less to the subjects in the text. In the future, perhaps a better database to use would be one that could not be considered political in any nature. Additionally, another factor that could have led to a low accuracy score for our model is the range of our data. The majority of our data fell between the 2-3 range, which perhaps could have skewed our model. In the future, it might be better to increase the range of our labels from 1-10 to represent a wider range of compassion levels to yield better results. However, this could also make it harder for annotators to come to an agreement on the label of a text.