1 Regression

1.1 Introduction

In the previous two chapters we attempted to construct generic measures that may work for every graph, but in the end certainly did not perform on this graph. In this chapter we will construct a measure specifically for this dataset. Towards this end, we will use elastic net regression to learn what it means for s to be less than t. We construct a variety of feature representations using adverbs and/or phrases that co-occur with the adjectives for this task, and divide our annotated data into train, validation, and test sets to assess their efficacy.

1.2 Data Set

Recall we have three sources of comparisons, N-gram data alone, PPDB data alone, and the combination of N-gram and PPDB data. For each data set, we extracted the pairs of adjectives where no data exists in corpus. The number of pairs for each annotated set for each data set is displayed in table 1. We will learn a different model for each data set (N-gram, PPDB, PPDB + N-gram) and assess their efficacy using their respective validation and test sets.

1.3 Literature Review

Note in our graph we have 610 adverbs and phrases, and since we are interested in representing an adjective using its co-occurrence with adverbs/phrases, the feature representation could be large than the number of examples. Furthermore, this representation will be very sparse because most adjectives do not co-occur with most adverbs/phrases at all. Thus regularization will be necessary to prevent over-fitting. In this section we will give a brief overview of elastic net regression. Suppose our

data set is $(\boldsymbol{X}, \boldsymbol{y})$ so that $\boldsymbol{X}$ is the $n \times p$ design matrix, where each input x is represented by the appropriate feature vector $\boldsymbol{x} \in \mathbb{R}^p$, and let $\boldsymbol{y}$ be the n-dimensional response vector $\boldsymbol{y} = (y_1, \ldots, y_n)$. We assume $\boldsymbol{y}$ is generated by this process:

$$\boldsymbol{y} = \boldsymbol{X}\beta + z,$$

where z is a zero mean Gaussian noise factor. Then elastic net regression will recover the estimated $\widehat{\beta}$ where:

$$\widehat{\beta} = \arg\min_{\beta} \|\boldsymbol{y} - \boldsymbol{X}\beta\| + \lambda_2 \|\beta\|_2^2 + \lambda_1 \|\beta\|_1 .$$

Roughly, the l_1 penalty encourages a sparse solution where only a few variables in $\boldsymbol{x}$ participate in predicting $\boldsymbol{y}$, while the l_2 penalty encourages "grouping" so that more than a few variables in $\boldsymbol{x}$ participates.

1.4 Problem Formulation

In our setting, we will define:

$$y = \begin{cases} 1 & s < t \\ 0 & otherwise. \end{cases}$$

And for each pair of adjectives s and t. Additionally, we will need to find a corresponding feature representation so that:

$$\boldsymbol{x} = g(\phi(s), \phi(t)),$$

for some function ϕ and $g : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^p$, where $m \leq p$. In a departure from notation of previous chapter, s now refers to the string representation of the word, while $\phi(s)$ is the corresponding vector representation. If we have this model $\widehat{\beta}$ and the appropriate g and ϕ, then we can use this definition.

Definition 1.1. Given words s and t, and their representation $\boldsymbol{x} = g(\phi(s), \phi(t))$, and let:

$$\hat{y} = \widehat{\beta}'\boldsymbol{x},$$

then we can define:

$$\boldsymbol{Pr}[s < x] = \begin{cases} \frac{1}{2} + \epsilon & \hat{y} < \delta \\ \frac{1}{2} - \epsilon & otherwise, \end{cases}$$

for an appropriate threshold δ. Again we discard the value of $\hat{y}$ and define the probability by fiat.

1.5 Feature Representations

In this section we describe two broad sets of features we use, and their associated function g. In the first set of features, we will represent the adjective by the frequency of adverbs incident and/or outgoing from this adjective. In the second set of features we will represent the adjective by adjectives that are its neighbors. In an attempt to avoid confusion, we will denote the first set of features $\phi(s)$, while the second set will be $\nu(s)$.

First we list the ϕ's.

1. In-neighbor only. So that for each adverb v:

$$\phi(s)_v^{in} = \begin{cases} n & \text{there are n edges pointing to } s \text{ from all neighbors with the adverb v} \\ 0 & otherwise. \end{cases}$$

2. Out-neighbor only. So that for each adverb v:

$$\phi(s)_v^{out} = \begin{cases} n & \text{there are n edges pointing from } s \text{ to all neighbors with the adverb v} \\ 0 & otherwise. \end{cases}$$

3. Concatenation of in and out neighbor. So that $\phi(s) = (\phi^{in}, \phi^{out})$. In this case we also considered $\phi(s) = (\phi^{in}, -\phi^{out})$, where $-\phi^{out}$ is scalar multiplication of -1 with all entries of ϕ^{out}.

4. Element wise addition of in and out neighbor. So that $\phi(s) = \phi^{in} + \phi^{out}$.

5. Element wise subtraction of in and out neighbor. So that $\phi(s) = \phi^{in} - \phi^{out}$.

Furthermore, for each ϕ^{in} and ϕ^{out}, we can vary the number of adverbs in the vector. We sort the adverbs by frequency of appearance and pick the top 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, and all 605 adverbs/phrases.

Now we consider the ν's. In this case we find all neighbors of s and represent each neighbor by the frequency of adverbs between the neighbor and our vertex.

1. In-neighbor only. So that for each neighbor t:

$$\nu(s)_t^{in} = \begin{cases} n & \text{there are n edges pointing to } s \text{ from } t \\ 0 & otherwise. \end{cases}$$

2. Out-neighbor only. So that for each neighbor t:

$$\nu(s)_t^{out} = \begin{cases} n & \text{there are n edges pointing from } s \text{ to } t \\ 0 & otherwise. \end{cases}$$

3. Concatenation of in and out neighbor. So that $\nu(s) = (\nu^{in}, \nu^{out})$. Again we also experiment with $\nu(s) = (\nu^{in}, -\nu^{out})$.

4. Element wise addition of in and out neighbor. So that $\phi(s) = \phi^{in} + \phi^{out}$.

5. Element wise subtraction of in and out neighbor. So that $\phi(s) = \phi^{in} - \phi^{out}$.

6. Bernoulli representation of in and out neighbor. So that for each neighbor t we have:

$$\nu(s)_t = \begin{cases} \frac{|\{(t,s)\in \boldsymbol{E}\}|}{|\{(t,s)\in \boldsymbol{E}\}|+|\{(s,t)\in \boldsymbol{E}\}|} & \text{there is at least one edge from } t \text{ to } s \\ \frac{1}{10} & otherwise. \end{cases}$$

In all cases, we sort the neighbors so that the most connected neighbor to s is at the top, and so on. Finally we take the top n neighbors of s as its feature representation, where n is also an experimental parameter. If an adjective has less than n neighbors, then we pad the vector with 0. Note how we smooth out the definition of ν above so that the padding can be distinguished from a case where there are vertices from s to t, but not vice versa.

Finally, we explore a variety of g's:

1. element wise addition. $\boldsymbol{x} = \phi(s) + \phi(t)$,
2. element wise subtraction. $\boldsymbol{x} = \phi(s) - \phi(t)$,
3. concatenation. $\boldsymbol{x} = (\phi(s), \phi(t))$,
4. dot product. $\boldsymbol{x} = \phi(s) \cdot \phi(t)$.

After g has been applied, we also experimented with normalizing the entries of ϕ so they are between 0 and 1.

1.6 Results

In addition to varying the feature representations, we also varied the emphasis over each of the two penalty terms in the objective function. All results are displayed in the tables below. Before a more nuanced discussion, we will highlight a few points:

	Mohit	Turk	BCS
N-gram	550	586	556
PPDB	750	182	294
PPDB + N-gram	408	170	290

Table 1: The base-comparative-superlative pairs form the training set for each data set, the Turk pairs form the validations set, while Mohit's pairs will be the test set. Note in almost all cases but one, the test set is actually larger than than the training set.

	N-gram			PPDB			PPDB + N-gram		
Test set	**Pairwise**	**Avg.** τ	**Avg.** $\lvert\tau\rvert$	**Pairwise**	**Avg.** τ	**Avg.** $\lvert\tau\rvert$	**Pairwise**	**Avg.** τ	**Avg.** $\lvert\tau\rvert$
Mohit	00.0%	0.00	0.00	00.0%	0.00	0.00	00.0%	0.00	0.00
Turk	00.0%	0.00	0.00	00.0%	0.00	0.00	00.0%	0.00	0.00
BCS	00.0%	0.00	0.00	00.0%	0.00	0.00	00.0%	0.00	0.00

Table 2:

- Once again none of the methods passed the $\log(n)$ threshold we wished. In fact, no method performed above 71% accuracy.
-