W.P.I. D.S.501; Introduction to Datascience

To: Dr R. Paffenroth, Dr X. Kong
From: Matt Curcio
Due Date: December, 2014
Subject: Case Study #3

Using a publicly available movie database, www.cs.cornell.edu/people/pabo/movie-review-data/review_polarity.tar.gz. I will investigate movies ratings from a newsgroup. The ratings are not categorized or rated one through five stars but are reviews written by contributors to the newsgroup. I will implement a set of machine learning algorithms in the hope of teasing out good and bad movie reviews. The machine learning algorithms include tfidf vectorizers, k-nearest neighbors, and support vector machine learning.

First let us discuss how machine learning is broken down pedagogically. The first set of terms that I will discuss is whether the learning process is either supervised or un-supervised. Either manner data is compiled in such a way that it will return a solution that is inherent in the system, while sometimes no prior assumptions are made. Supervised learning is when the independent or explanatory variables used as input for the learning system have a dependent outcome, predictor or target value. Un-supervised learning refers to the fact that the independent variables used as input do not have a specific predictor to draw conclusions from.

An other way to break down machine learning is to develop the ideas of clustering, correlation or regression. Clustering is generally thought of as an unsupervised learning technique but can be supervised as well. Clustering is a technique for finding similarity among a member of groups. Groupings are identified by a set of independent variables and using some distance measurement articles are clustered together by how close its members are to a centroid. There are many different measurement equations that are commonly used. The quality of a clustering result depends on the overall algorithm, the distance function chosen, and its application. K-Nearest Neighbors is considered a clustering technique and will be used in this analysis. In general, the methods used in this report fall under the umbrella of unsupervised learning and they include Term Frequency–Inverse Document Frequency (TFIDF) and support vector machines (SVM).

TFIDF - Term Frequency–Inverse Document Frequency

There are 2000 samples that were used from within this dataset. The first task which was undertaken was to do a term frequency–inverse document frequency. Term frequency is a measure of the number times a term is counted in a document.

$$Eq.1 \qquad tf(t,d) = 0.5 + \frac{0.5\ x\ f(t,d)}{max\{f(w,d) : w \in d\}}$$

in simpler terms term frequency can be looked at as simply:

tf (t) = (Number of times term t appears in a document) / (Total number of terms in the document)

Inverse document frequency uses two fairly simple sets of algorithms to assign a numerical value on terms of interest.

$$Eq.2 \quad idf(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

While the *idf* can be thought of as a measure of:

idf (t) = log (Total number of documents / Number of documents with term t in it)

The term frequency (tf) normalizes the terms in a document. The term now becomes a probability within the document and in the entire corpus of documents. This normalization will weed out documents that use a term with low probability. The spontaneous use of a low frequency, unusual words (such as onamonapia) will not divert a word search to that document because its incidence even in that document may be very low. Additionally, the log score of the term frequency will be very close to zero. The log function tends to 'iron out' the ripples that might be found in a term frequency vector. For a word to be 'meaningful', in the sense that it is a good match for a search term, the word in question must be used consistently through the document.

Investigating the confusion matrix for the TFIDF calculations show a good overall trend for picking up the positive and negatives terms which are in the newsgroup posts. The diagonals show or concordant pairs show a good correlation.

Figure #1: TfidfVectorizer confusion matrix

[mean: 0.83533, std: 0.01221, params: {}]

	precision	recall	f1-score	support
neg	0.85	0.84	0.85	248
pos	0.84	0.86	0.85	252
avg / total	0.85	0.85	0.85	500

[[208 40]
[36 216]]

Figure #2: Best parameters for TfidfVectorizer:

vect__max_df: 0.87
vect__min_df: 2

	precision	recall	f1-score	support
neg	0.85	0.86	0.85	248
pos	0.86	0.85	0.85	252
avg / total	0.85	0.85	0.85	500

[[213 35]
[38 214]]

N-Grams

N-grams are another common method of searching for terms in a document. They also use a probabilistic underpinning for determining which terms might be useful to a searcher. It is interesting to me the N-grams are used in bio-informatics.

This is due to the fact that DNA and protein structures have a tendency to use motifs over and over again. Repetition is a common evolutionary practice to use domains that perform a certain and

in a sense time tested function. These motifs are commonly called domains most likely due to the mathematical term that describes a connected set of points or using the language of probability the set of values which an independent variable may take.

Continuing, a protein domain may be used many times in the course evolutionarily related proteins. Therefore these domains have similar amino acid residues in a sequence they can be sought through out other species genomes or even with in a species.

The N-gram searches for word groupings in a specific manner. For example, the bi-gram seeks out 'word' pairs of the size 2 and all combinations there in to find all combinations of 20 amino acid residues. N-grams also are a function or relative frequency as are TFIDF with some differences. The general form of N-gram calculations is:

$$Eq.3 \qquad \hat{p}(word_a) = \frac{count(word_a)}{Total\ number\ of\ words\ per\ training\ set}$$

$$Eq.4 \qquad \hat{p}(word_b | word_a) = \frac{count(word_b, word_a)}{\sum_{word_b} count(word_b, word_a)}$$

The equation four gives a probabilistic value for the set of words that the computer is trained on. A vector is produced that contains ONLY words in the training set. This is a major short coming for the N-gram approach. Initially if your word of interest is not in your initial training set the probability will be zero. Therefore this will preclude finding words or word N-grams that might use that word in the future testing sets.

Subsequently, the second mathematical statement now searches for (in this circumstance) word pairs. The algorithm calculates the probability that word a will be next to word b and proceeds to count these values in the entire corpus. Again as seen before if the first word is not seen then all subsequent word N-grams will be scored very low (zero) and not returned by the search function.

Considering the case where the first word is in your training set the program will now search for words proximate to each other. Here again, if a word N-gram is not seem in the training set, it will not be returned as a match. Since the number of possibilities can be very high for a large vocabulary the matrix it produces for N-grams will be N-squared. So for example, if a document has 30,000 words in it the number of bi-grams will produce a matrix of 30,000 squared or 9E8. This would require tremendous computing power and time to go through the entire experimental space.

As stated before, or at least implied, N-grams are useful in finding DNA or protein domains since what one is looking for will be taught via the training. However, there are even difficulties in this circumstance. One variations in a DNA code can render the exact matching capability useless since this will also signify a very low probability. This exemplifies the inability of this type of machine learning to deal with very minor changes in language. Conversely, since DNA and Protein sequences are not a formal language the idea of simply seeking phrases in the form of DNA (ATCG) and 20 amino acid residues letters is very useful.

Figure #3: Ngram-range:
Best parameters for TfidfVectorizer:

vect__ngram_range: (1, 2)

	precision	recall	f1-score	support
neg	0.87	0.85	0.86	248
pos	0.85	0.88	0.86	252
avg / total	0.86	0.86	0.86	500

[[210 38]
[31 221]]

Support Vector Machines & K-Nearest Neighbors

In this study there were two additional forms of machine learning. They include Support Vector Machines (SVM) and K-Nearest Neighbors (KNN).

K-nearest neighbors is a surprisingly simple idea. KNN can use the Euclidean distance of set of points, generally 2D, and minimizes the distance from a centroid. In the case of two dimensions it uses the form $Euclidean\ Distance(2D) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$. There are other aggregate formulas that can be used for continuous variables. However there are alternate forms that can be used in place of Euclidean distance and for categorical variables. One such expression is known as the Hamming distance, $D_H = \sum_{i=1}^{k} |x_i - y_i|$. This is useful in the case of dummy or indicator variables that might be seen in binary data for example. An other interesting form that is used with respect to KNN is a weighted Euclidean distance. $Weighted\ Distance = \sqrt{\sum_{i=1}^{n} w_i^2 (x_i - y_i)^2}$. The minimization process can utilize mean square error (MSE).

Lastly, support vector machine (SVM) learning uses maximum margin classification. Conceptually the algorithm computes a line that bisects the groups in question. This bisection is called the decision boundary. The boundary is chosen such that a hyperplane is equidistant from the two nearest points. Where as ordinary least squares minimizes the distance along the x axis the SVM minimizes the distance along the hyperplane which does not need to be along a conventional x,y,z (or high dimension) axis.

Figure #4: Machine Learning Algorithms
Best parameters for LinearSVC:

clf__C: 1001

	precision	recall	f1-score	support
neg	0.85	0.86	0.86	248
pos	0.86	0.85	0.86	252
avg / total	0.86	0.86	0.86	500

[[214 34]
[37 215]]

Figure #5: Machine learning algorithms,
Best parameters for KNN:

clf__n_neighbors: 10
clf__weights: 'uniform'

	precision	recall	f1-score	support
neg	0.73	0.63	0.68	248
pos	0.68	0.77	0.72	252
avg / total	0.70	0.70	0.70	500

[[156 92]
[58 194]]

Examples of False Positives:

joe versus the volcano is really one of the worse movies made in very recent memory .
the strangest thing is that you would think nothing would go wrong with it .
it has a solid cast with tom hanks & meg ryan as the lead roles .
but you can never judge a movie by its cast . . .
if there is one good thing about joe vs the volcano , it is that the plot is original .
unfortunately , it is also incredibly stupid .
the movie begins with joe (tom hanks) going to work .
this opening sequence is very boring and slow .
it shows joe walking to his office .
but on the way , he has to wait in a long line passing by strange and slightly depressing scenery with obnoxious lighting

Examples of False Negatives:

synopsis : it's 1977 in a small italian american neighborhood in new york .
disco music is popular , and the sexual revolution is on , but the people of new york are not happy .
in fact , they're terrified .
as in spike lee's do the right thing a heat wave has the city in its grip , and peoples' tempers are flaring .
the cool of night offers no respite : the neighborhood is being terrorized by a nocturnal psychotic killer who calls himself the son of sam .

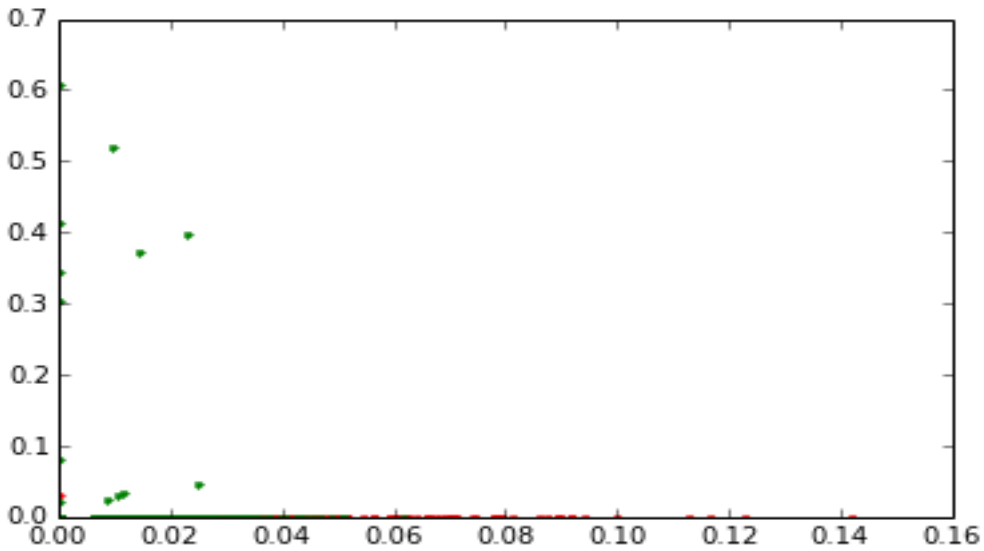

Figure #6:

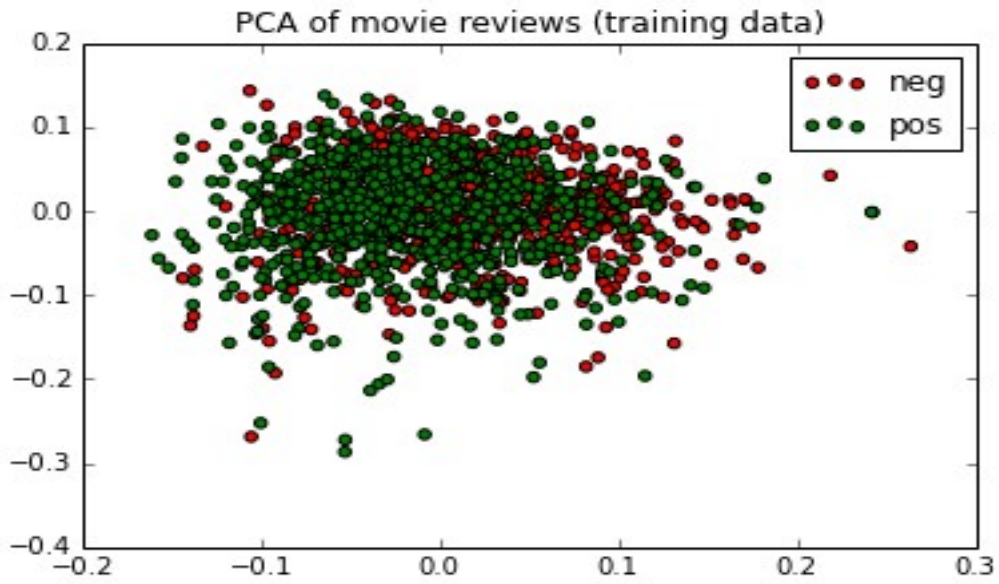

Figure #7:

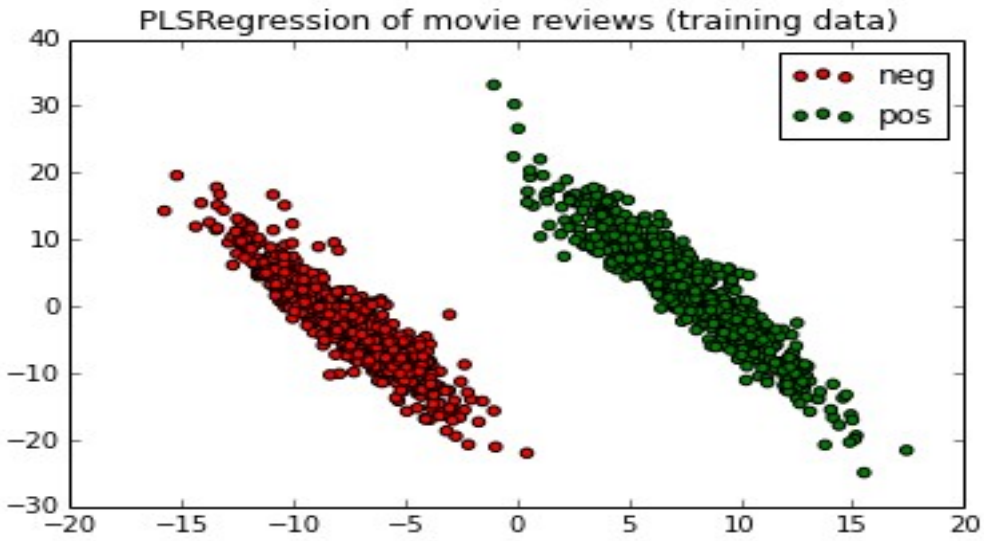

Figure #8:

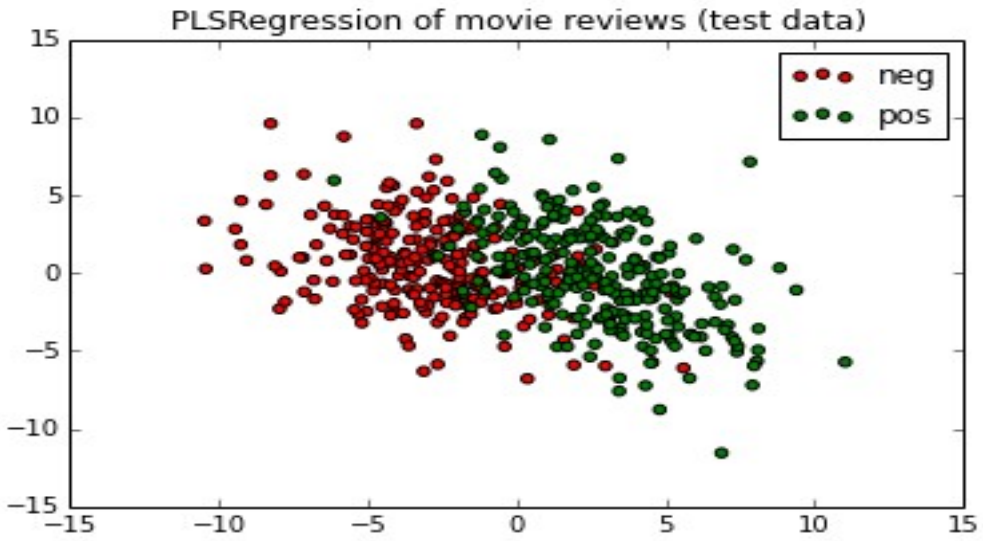
PLSRegression of movie reviews (test data)
neg
pos
15
10
5
0
−5
−10
−15
−15
−10
−5
0
5
10
15

Figure #9: