Linear and Logistic Analysis on User Reviews from Yelp

Classification and regression applied to bag-of-word consumer reviews using gradient descent and comparative methods with feature selection

Abstract

In this document, the authors first present the methods used to analyse sparse and large amount of data, specifically a bag-of-words data set obtained from Yelp user reviews. Among these methods are included logistic discrimination with gradient descent, naïve Bayes and decision trees for classification (*Methods classification)*; and multivariate logistic, linear and multiple linear methods for regression (*Methods regression)*. Next, a number of models were trained using these methods, while including feature selection for some of them. These steps are described more closely under the *Experiments* section. Once the models were trained, they were used to predict whether a user will leave more than three stars in his/her review (classification) and how useful other users find the review (regression). Lastly, the performance of the models is presented and analysed in the *Discussion* section. Sample code of the methods can be found in the *Appendix* section.

Table of Contents

Introduction

This project has extended and completed our understanding of the machine learning topics addressed during the course and outside of it. We hoped to learn more on practical applications of the methods under discussion by drafting them on our own. This project presented an excellent learning opportunity in that regard, which can be said to have been fulfilled from our own perspective.

In this project, we addressed the question of whether iterative methods, such as gradient descent, compare to analytical or otherwise more deterministic methods. The bag-of-word data used operates as an example and a fair point of comparison to answer this question.

As another part, we address the significance of feature selection for these methods, ranging from the utilization of a single best feature to combining all of them. This raises the key concept of determining the extent of the required features for representativeness of the data set.

Addressing these questions provides more clarity for practitioners of machine learning on deciding the relevant methods for the given data as well as to the relevance of their features.

Methods

Classification

Gradient Descent and Logistic Discrimination

For classification, gradient descent is an iterative optimization method for minimizing the classification error, E(w|X), with w denoting the set of parameters on the given training set X[Alpaydin, p. 219].
To accomplish this task, gradient descent utilizes the gradient vector composed of the partial derivatives for the error over each weight. Starting from a random or fixed w, the procedure minimizes E at each step updating w as presented in (1), where η denotes the stepsize of each iteration.

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}, \forall i \quad (1)$$

[Alpaydin, p. 219]

This gradient is added as an update to the existing weights (equation 2).

$$w_i = w_i + \Delta w_i \quad (2)$$

[Alpaydin, p. 219]

When the update is below a given threshold value, the iteration process is terminated and the resulting weights are returned to the classifier.

To utilize this in logistic discrimination, the weights were used to determine the higher probability class for the incoming inputs in order to classify them to the two classes.

The basis for choosing this method came from our experience with it during the course, as well as the promise it held for larger data sets, for which an analytical solution could be non-trivial to determine.

Naive Bayesian

To simplify linear discriminant, the variables can be assumed to be independent for a given class C. The resulting classifier is called the naive Bayes' classifier[Alpaydin, p. 97]. This classifier is a simple, accessible method which can give results or at least serve as a decent reference point or "base benchmark". The way to implement it is described below.

Additionally, since the present case is a binary classification (0 or 1), the prior for a class can be determined with equation (3).

(3) $$P(C=0) = a \text{ and } P(C=1) = 1 - a$$

[MLBP, lecture 5]

Next, both conditional probabilities $P(x|C_i)$ are modelled as two separate Bernoulli distributions, and each one is independently learned from the data observations belonging to their own class, formally presented in (4).

(4) $$P(x|C_i) = \prod_{j=1}^{d} p_{ij}^{xj} (1-p_{ij})^{1-xj}$$

[MLBP, lectures 5,7]

In the present case, the following modifications were carried on. First, the maximum likelihood estimate for p_{ij} was defined as presented in (5), where the alpha term was added in order to avoid zero or unit probabilities, K indicates the number of classes, r^t indicates a class and x_j^t is data belonging to that class.

(5) $$p_{ij} = \frac{\alpha + \sum_t x_j^t r_i^t}{K\alpha + \sum_t r_i^t}$$

[MLBP, exercise 5]

Second, the weights w_0 and w_j are given by the relations presented in (6).

(6) $$w_o = \log\frac{1 - p_r}{p_r} + \sum_{j=1}^{d} \log\frac{1 - p_{0j}}{1 - p_{1j}}$$

$$w_j = \log\frac{p_{0j}(1 - p_{1j})}{p_{1j}(1 - p_{0j})}$$

[MLBP, exercise 5]

The relations in (6) are obtained by treating the equation $y = w'x + w_o$ as a switched sigmoid. Also from (6), p_r is the prior probability defined as shown in (7).

$$p_r = \frac{\sum_{t=1}^{N} r^t}{N} \quad (7)$$

The present method was chosen because of its relatively easiness to implement and because its assumption that each class has a different distribution from the other. In other words, it is assumed that a comment from a person leaving 3 or less stars has a different distribution over the bag of words than the comment from a person leaving more than 3 stars in his/her review.

Decision trees

As described in the course book: “A decision tree is a hierarchical data structure implementing the divide-and-conquer strategy.” In this case, this hierarchy comes from dividing the input data into local regions based on a distance measure such as Euclidean norm[Alpaydin, p.185]. These local regions aim to represent simpler and more similar data. The decision trees are a nonparametric approach as they do not require initial parameters, rather learn the parameters and the underlying model for the future data from the training data.

Each division produces either a decision or a leaf node. When the resulting region represent a single output clearly enough, the current branch concludes with a leaf node. Otherwise, more divisions are required, and hence, another decision node is created to reduce the diversity.

The iteration at each branch is continued until a leaf node is formed. However, in the approach of this project, the level of the tree acted as another concluding factor. To clarify, if a leaf node was not formed by the nth layer of decisions, the branch was forcefully stopped. This avoided overcomplexity with lesser impactful decision after another, leading into overfitting of the parameters and the model.

Decision trees have some internally implemented feature selection[Alpaydin, p. 197]. This was crucial with the several features used for determining the output. The method gave clarity on importance of the features, allowing efficient predictions with less dimensions and complexity.

Regression

Multivariate linear regression with gradient descent

Similarly, as presented in *Classification* section, the gradient descent algorithm can also be utilized with regression. Equation (8) represents the update of the model parameters, where the alpha denotes the learning rate, m the number of samples, h for hypothesis output based on parameters and y as the class result[Holehouse].

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)} \quad (8)$$

[Holehouse]

It becomes apparent that the iteration is terminated as the hypothesis predicts the class output with a sufficient accuracy.

Linear regression with gradient descent allowed the comparison of features and their relative significance to predicting the correct output. Instead of choosing just one feature or all of them, this method allowed combination of several features which were determined relevant for the predictions. This resulted in reduction to computation for the future data as only the most relevant features (based on training) were considered. In addition, the dimensionality reduction that was formed to be part of this process results in less overfitting for the models.

Linear regression

In this method, the model is a linear function of the input with slope w_1 and intercept w_0. The model can be written as presented in (9).

$$g(x^t | w_1, w_0) = w_1 x^t + w_0 \quad (9)$$

[Alpaydin, p. 74]

The parameters w_1 and w_0 are learnt from the data as explained below.

First, let us introduce an error function as shown in (10).

$$E(\theta|X) = \frac{1}{2}\sum_{t=1}^{N}[r^t - g(x^t|\theta)]^2 \quad (10)$$

[Alpaydin, p. 74]

The error function should be minimized with respect to the parameters θ, called least squares estimates. To find the parameters θ, first the derivative of the error function is taken with respect to w_0 and w_1, obtaining (11).

$$\sum_t r^t = Nw_0 + w_1\sum_t x^t$$
$$\sum_t r^t x^t = w_0\sum_t x^t + w_1\sum_t (x^t)^2 \quad (11)$$

[Alpaydin, p. 75]

Which can be written in matrix form as Aw = y, from which the parameters w can be solved by $w = A^{-1}y$. The terms of the equation are presented in (12).

$$A = \begin{bmatrix} N & \sum_t x^t \\ \sum_t x^t & \sum_t (x^t)^2 \end{bmatrix}, \quad w = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}, \quad y = \begin{bmatrix} \sum_t r^t \\ \sum_t r^t x^t \end{bmatrix} \quad (12)$$

[Alpaydin, p. 75]

For this method, no other modifications were implemented.

Linear regression, similarly to naïve Bayes for classification, was chosen as a starting point and base benchmark. Even more, an online article from FastML[suggested that for sparse, high-dimensional data (such as bag-of-words) a linear model should be used.

[FastML]

Multiple linear regression

Multiple linear regression is very similar to the previous method, as the output is a linear function (weighted sum) of the input, which in this case is a series of variables. Equation (13) describes the previous affirmation in a more formal notation.

$$r^t = g(x^t|w_0, w_1 \dots, w_d) = w_0 + w_1 x_1^t + w_2 x_2^t + \cdots + w_d x_d^t \tag{13}$$

For this method, the error function is presented in (14).

$$E(w_0, w_1, \dots, w_d|X) = \frac{1}{2}\sum_t (r^t - w_0 - w_1 x_1^t - w_2 x_2^t - \cdots - w_d x_d^t)^2 \tag{14}$$

From equation (14), the derivatives are taken with respect to the parameters w_j, j=0…d, obtaining equations presented in (15).

$$\begin{gathered}
\sum_t r^t = Nw_0 + w_1 \sum_t x_1^t + w_2 \sum_t x_2^t + \cdots + w_d \sum_t x_d^t \\
\sum_t x_1^t r^t = w_0 \sum_t x_1^t + w_1 \sum_t (x_1^t)^2 + w_2 \sum_t x_1^t x_2^t + \cdots + w_d \sum_t x_1^t x_d^t \\
\sum_t x_2^t r^t = w_0 \sum_t x_2^t + w_1 \sum_t x_1^t x_2^t + w_2 \sum_t (x_2^t)^2 + \cdots + w_d \sum_t x_2^t x_d^t \\
\vdots \\
\sum_t r_1^t x^t = w_0 \sum_t x_1^t + w_1 \sum_t (x_1^t)^2 + w_2 \sum_t x_1^t x_2^t + \cdots + w_d \sum_t x_1^t x_d^t
\end{gathered} \tag{15}$$

Which again can be put in matrix format as shown in (16).

$$A = \begin{bmatrix} 1 & x_1^1 & x_2^1 & \cdots & x_d^1 \\ 1 & x_1^2 & x_2^2 & \cdots & x_d^2 \\ \vdots & & & & \\ 1 & x_1^N & x_2^N & \cdots & x_d^N \end{bmatrix}, \quad w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}, \quad r = \begin{bmatrix} r^1 \\ r^2 \\ \vdots \\ r^N \end{bmatrix} \tag{16}$$

The normal equations are: presented in (17) and the parameters w can be solved as shown in (18).

(17) $$W^T X w = X^T r$$

(18) $$w = (X^T X)^{-1} X^T r$$

[Alpaydin, p. 104-105]

Following the steps from single linear regression, multiple linear regression was tried as well in an attempt to improve the accuracy on the test data. This method allowed better understanding for the relevance of the feature selection process, by combining several to all of these features for forming multiple linear regression.

Experiments

Classification

For different methods, different pre-processing steps were followed. For some, only few basic steps were applied, common to all our approaches. The common steps, include dividing the given training set into a feature or design matrix and an output column. The following discusses each method and the experimentation with them in detail.

In the case of the gradient descent method in this project, the design matrix was normalized to give an equal opportunity for every feature to determine the weights. Similarly, the weights were initialized to zeros to obtain consistent results. The maximum number of iterations was set to 4000 and the termination condition with the gradient was held below 1e-5. The source code, for Matlab, can be found in Appendix 1. With the preparations done, these weights were calculated using the training data for the gradient descent. The plot of the weights is presented in Figure 1.

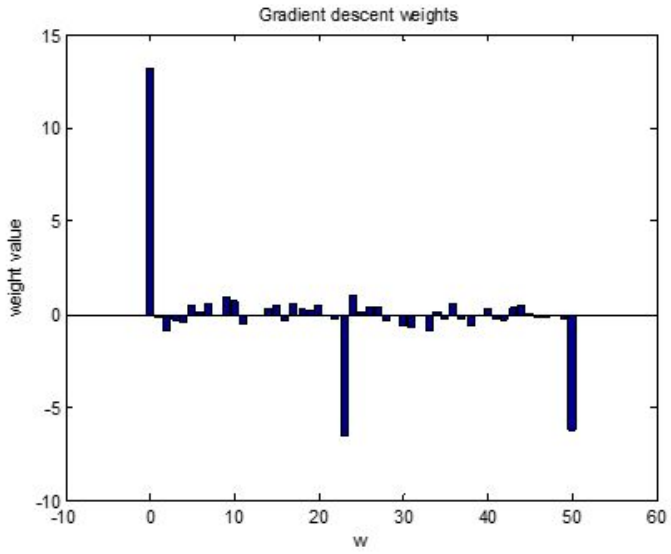

Figure 1. Gradient descent weights.

From Figure 1, it can be extracted that certain features, specifically 23 and 50, have the highest values among all features. Based on this information, a second prediction was carried out by using only these two weights, in addition to the bias weight at position 0 in the graph, on the test data. The results are presented in the *Results* section.

For naïve Bayes, no further data pre-processing was carried on. The source code can be

found in Appendix 2. The weights are presented in Figure 2.

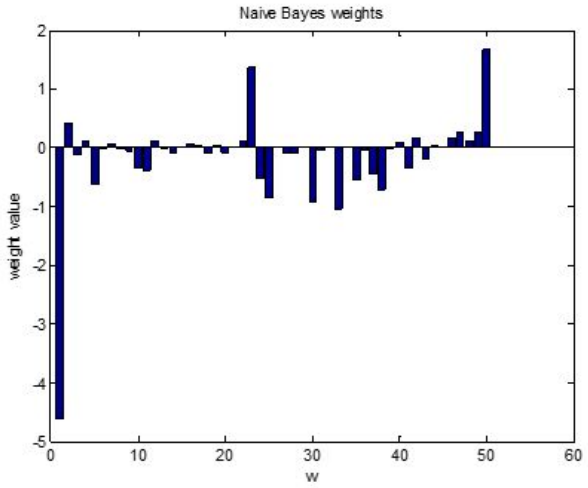

Figure 2. Naïve Bayes weights.

In an attempt to verify that the value chosen for the alpha was correct, a second version of Naïve Bayes was implemented for a range of alphas (0 up to 9000) and even cross-validation (k-folds) was applied. This attempt, however, was unsuccessful as the average error decreased up to a limit, after which it stabilized as can be seen in Figure 4.

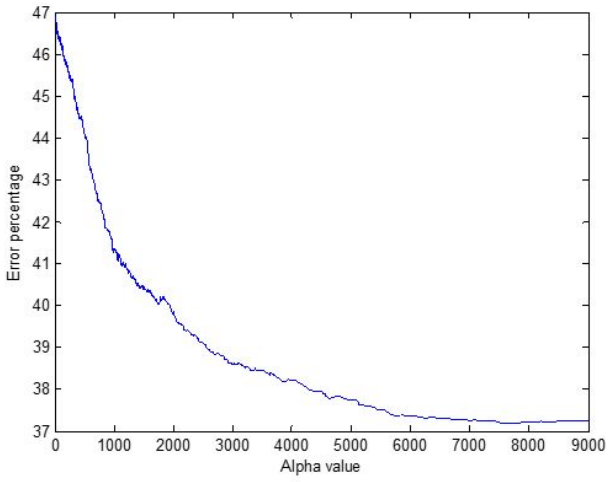

Figure 4. Average error for different alpha values and using Naive Bayes.

Decision trees were implemented using *fitctree* and *predict* commands from Matlab's Statistics and Machine Learning Toolbox[Matlab Stat]. First, the classification tree was constructed using *fitctree with* the training data. Then, this tree was used to predict the output of the test parameters given to it.

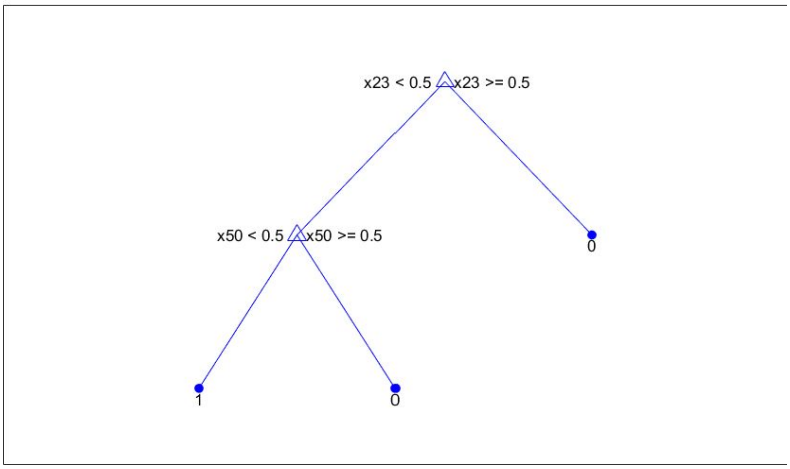

Figure 4. The best classification decision tree with pruning level of 50, hence leaving only two layers. This tree provided 71.8% accuracy with the test data.

After testing out different options for tuning the decision tree obtained from *fitctree,* limiting the number of decision nodes or pruning[Alpaydin, p. 194] gave the most immediate rewards in terms of improving the test accuracy (see Figure 4). The test accuracy results for various pruning levels are presented in a table at the *Results* section.

Regression

In the case of regression, the same basic pre-processing of data was applied as in the case of classification. Additionally, for the gradient descent method the data was normalized.

The gradient descent method was implemented from scratch and the source code can be found in Appendix 3. The parameters used for the maximum iterations was 2000 and the relative difference between the hypothesis and the class column was 1e-10. The latter being used to determine when to break off from the iteration step.

The output weights are presented in Figure 5. In this Figure, it can be concluded that at least 7 of the features clearly stand out (disregarding the bias term). For the second test, the class predictor was run on the test parameters, using only these 7 features, for which the results are presented in the *Results* section of the present report.

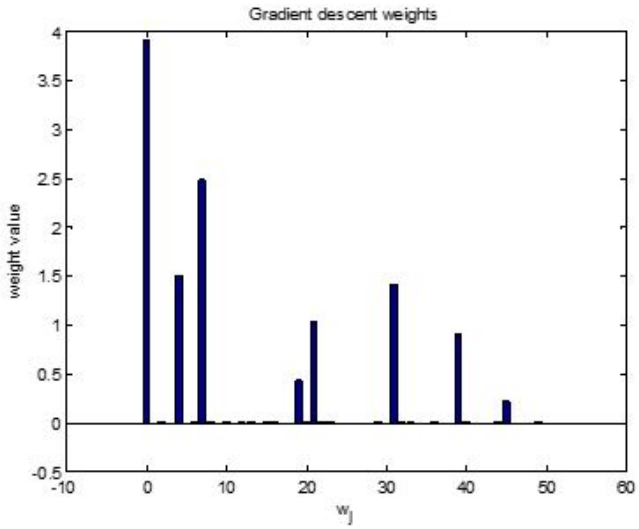

Figure 5. Gradient descent weights.

For the linear and multilinear regression methods two ready-made functions (*regress* and *fitlm*) were used from Matlab Statistics and Machine Learning Toolbox[Matlab Stat].

First, for the linear regression method the function *regress* was applied individually to each feature, obtaining a matrix with weights w_0 and w_1 (intercept and feature weight, respectively). The plot for both parameters for each of the 50 features is presented in Figure 6.

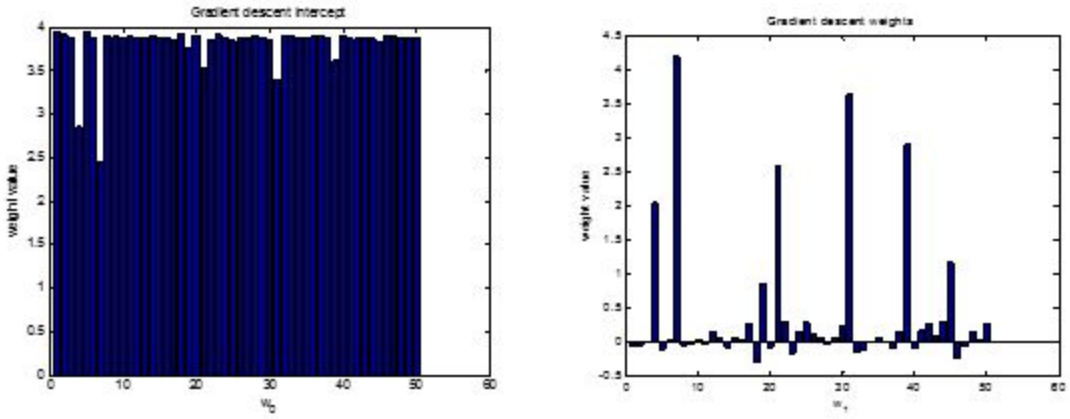

Figure 6. Intercept (left) and weights (right) for linear regression.

Second, for the multilinear regression method the ready-made function from Matlab called *fitlm* was used to fit a linear model using all the features at once. Since there are no parameters to change, no additional processing was required (e.g. cross-validation).

Results

Classification

Gradient Descent and Logistic Discrimination

Table 1: The accuracy of logistic discrimination

Features used	Accuracy training data (%)	Accuracy test data (%)
All	66.44	69.60
2 best (+bias term)	69.60	71.8

Naive Bayesian

Table 2: The accuracy of naive Bayesian

Features used	Accuracy training data (%)	Accuracy test data (%)
All	60.22	61.80

Decision tree

Table 3: The accuracy of decision trees

Pruning Level (prune(ctree,'Level', [x]))	Accuracy training data (%)	Accuracy test data (%)
0	89.0400	62.2
40	73.22	70.6
41	72.18	70.7
42	71.38	70.8
43	70.86	70.3
44	70.76	70.2
45	70.64	69.9
46	70.50	69.8

47	70.02	70.1
48	68.74	71.4
49	68.56	71.7
50	**68.14**	**71.8**
51	66.62	70.2
52	62.86	62.80

Regression

Gradient descent and Linear Regression

Table 4: The MSE of gradient descent and linear regression

Features used	MSE training data	MSE test data
All	0.0465	0.0475
7-9 best (+bias term)	0.0469	0.0472

Linear regression

In the case of linear regression applied for individual parameters the mean square errors (MSEs) were found to be rather high, hence the MSE for each parameter in particular were plot together. Figure 7 presents the MSE for the training data and Figure 8 for the test data.

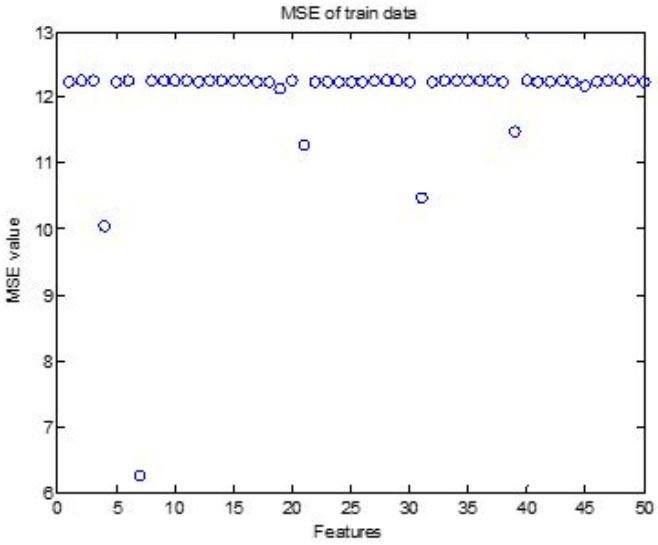

Figure 7. Mean Square Error on training data using linear regression method.

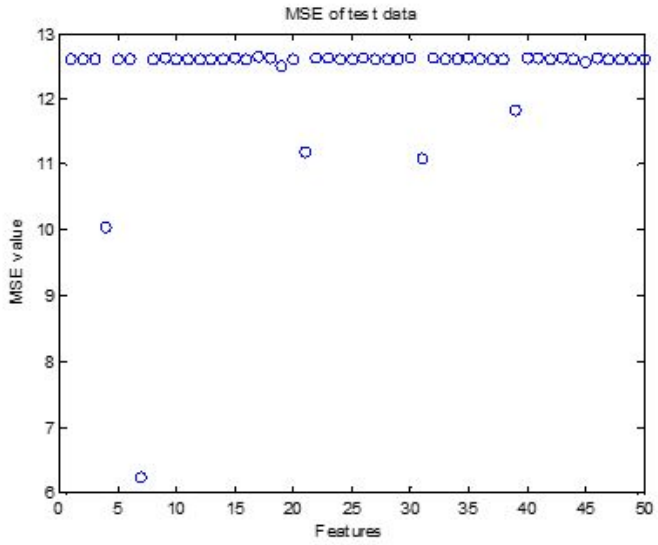

Figure 8. Mean Square Error on test data using linear regression method.

Multilinear regression

Table 5: The MSE of multilinear regression

Features used	MSE training data	MSE test data
All	0.0465	0.0475
5 best	0.2839	0.2619
4 best	1.0999	1.0750
3 best	2.1738	2.3089
2 best	4.0672	3.9405
1 best	6.2589	6.2259

Discussion

Classification

The three methods compared provide interesting insights into the data. Naive Bayesian, while simple to implement, provides an inaccurate model when using all of the features. Interestingly, this doesn't lead to low training error either. In comparison, the logistic discrimination method improves the result with all of the features in use (69.6% vs 61.8% for the test accuracy).

However, the logistic discrimination can be improved with feature selection. Using only the best two features, based on their weight values, the test accuracy increases to 71.8%. The basis for this change comes from the reduced overfitting to the training data. A similar trend can be seen with decision trees, in which the usage of pruning improve the result from 62.2% without pruning all the way to 71.8% using two most impactful features. The latter result being the same as with the logistic discrimination as well as the two features turn out to be the same ones.

Interestingly, these two most significant features were words 'never' and 'disappointing' which are both highly polarizing words. They were especially accurate in predicting the lower score when they appeared. Therefore, the results of this project are plausible from this perspective.

Based on these results, we can see that the feature selection can be an excellent tool to improving the prediction accuracy of the generated model. Another observation comes from the similarity of the test accuracy using logistic discrimination and decision trees with only two features. Therefore, the methods yield certain similarities, at least when less features are involved.

Regression

The regression task for the project utilized the frequency of 50 words in the user reviews as features to determine the number of votes from the users. To tackle this task, linear regression for each feature was calculated separately in respect to predicting the vote based solely on a single feature. For the majority of the features, the MSE was closer to 12.7. However, the best MSE of these single features was 6.2, which was later confirmed by multilinear regression method. The multilinear method presented more promise than linear method as it combined features together for a better approximation. When combining all of them, the result of 0.0475 was a significant improvement over previous results. However, not all features are equally important and they can even be detrimental to result by causing overfitting with the training data. To counteract this, only a few of features were combined for the slightly better MSE of 0.0472 as given by the multivariate gradient descent regression method. Moreover, the dimensionality of the original 50 features was reduced to seven of the most representative ones, which provides a significant improvement in the calculation time.

To conclude, no single feature was representative of the data, however, a combination of them presented better results. Furthermore, the optimal MSE values were obtained by using information from seven of the features. Hence, there was a difference in the feature selection between the regression and the classification, in which only two features were required.

Conclusion

This project answered the question of how to reduce dimensionality of a large and sparse data set. As well as presenting variations in methods for obtaining the output from that data. Iterative methods seemed to perform equally well with more deterministic methods, especially when less features were included. However, the weights values were affected based on the assumptions of the given methods. Hence, naive Bayes' assumption for independent variables corresponds to worse accuracy.

The best of the methods were also tested with Kaggle data challenge[Kaggle] set linked to this course. In comparison, the results were average which proves the availability of more effective methods for this task.

To determine improvements, the future methods could focus on forming new features by combining features as the words on their own are ambiguous, for example, using word 'never' can be used in both positive and negative contexts even though the latter is more common.

Another aspect to enhance the process could be to utilize a combination of multiple learners to capture more of the underlying distribution as they approach the output from different perspectives.

References

Author, *Title,* Publisher, Location, Date.

1. Alpaydin, E.
   Introduction to Machine Learning.
   MIT Press, Cambridge, England, 2010.
2. Holehouse.org: Stanford machine learning course lecture notes presented by Professor Andrew Ng.
   Linear Regression with Multiple Variables.
   http://www.holehouse.org/mlclass/04_Linear_Regression_with_multiple_variables.html
3. Kaggle Data Challenge
   https://www.kaggle.com
4. Machine Learning Basic Principles(MLBP) course lectures 2016.
   https://mycourses.aalto.fi/course/view.php?id=13034
5. *Matlab Statistics and Machine Learning Toolbox*
   Documentation available at:
   https://se.mathworks.com/help/stats/index.html?s_cid=doc_ftr
6. Zygmunt Z.
   Classifying test with bag-of-words: a tutorial.
   Available at: http://fastml.com/classifying-text-with-bag-of-words-a-tutorial
   FastML, 2015.

Appendices

APPENDIX 1: Logistic discrimination with gradient descent function in Matlab

```
function [w_out,flag] = gradient_descent(param,class,w_0,alpha)
      % Initialize variables
      w = w_0;
      maxIter = 4000;
      relDiff = 1e-5;
      i = 0;
      flag = 0;

      while(1)
            % Make prediction using current weights
            pred = 1 ./ (1 + exp(-param * w));
            % Calculate gradient as simple rest between real class and
            %prediction
            grad = ((class - pred)' * param)';
            % Update current guess
            w = w + alpha * grad;

            % Check for convergence
            if abs(grad) < relDiff
                  break;
            end
            if i > maxIter
                  flag = 1;
                  break;
            end
            w_out = w;
            i = i + 1;
      end
end
```

APPENDIX 2: Naive Bayesian Matlab source code

```
%% Start clean
clear all

%% First load training data
train_data = csvread('classification_dataset_training.csv',1);

% Stripe down the parameters and class of the training data
train_param = train_data(:,2:51);
train_class = train_data(:,52);
[N,f] = size(train_param);

% Calculate parameters w
alpha = 1;
K = 2;
p_r = sum(train_class) / N;
p_0j = zeros(1,f);
p_1j = zeros(1,f);
for i=1:f
      p_0j(i) = (alpha + sum(train_param(~logical(train_class),i))) / (K * alpha +...
      sum(~logical(train_class)));
      p_1j(i) = (alpha + sum(train_param(logical(train_class),i))) / (K * alpha +...
      sum(logical(train_class)));
end

% Calculate weights w_j
w_j = log(p_0j .* (1 - p_1j) ./ (p_1j .* (1 - p_0j)));
w_0 = log((1 - p_r) / p_r) + sum(log((1 - p_0j) ./ (1 - p_1j)));
w = [w_0 w_j];

% Predict classes of training data and calculate classification errors
train_pred = 1 ./ (1 + exp(w_0 + train_param * w_j')) > 0.5;
train_error = (sum(train_pred ~= train_class) / N) * 100


figure
bar(1:size(train_data,2)-2, w_j)
xlabel('w');
ylabel('weight value');
title('Naive Bayes weights');

%% Test the (trained) classifier on the test data
test_data = csvread('classification_dataset_testing.csv',1);
test_data_sol = csvread('classification_dataset_testing_solution.csv',1);

stars_test = 1 ./ (1 + exp(w_0 + test_data(:,2:51) * w_j')) > 0.5;
% Get errors
test_error = (sum(stars_test ~= test_data_sol(:,2)) / size(test_data,1))*100
test_accuracy = 100 - test_error
```

APPENDIX 3: Multivariate linear regression with gradient descent in Matlab

```
w = zeros(51,1);
n_step = 0.001;
% Initialize variables
alpha = 0.1;
maxIter = 2000;
relDiff = 1e-10;
i = 0;
flag = 0;

while(1)
      % Make prediction using current weights
      train_pred = train_param * w;
      %train_pred = train_param * [w0; w];
      % Calculate gradient as simple rest between real class and prediction
      grad = ((train_pred - train_class)' * train_param)' ./ 5000;
      %grad0 = (train_pred - train_class)'./5000;
      % Update current guess
      w = w - alpha * grad;
      %w0 = w0 - alpha * grad0;
      % Check for convergence
      if abs(grad) < relDiff
      break;
      end
      if i > maxIter
      flag = 1;
      break;
      end
      i = i + 1;
end
```