Comparing Regularisation Techniques for Logistic Regression

Sujith Padaru & Pashupati Hegde

November 25, 2016

1 Abstract

In this report we tackle the problem of predicting rating for Yelp reviews based on the bag of words representation of the review. The problem is designed as classification task by considering reviews with ≥ 4 ratings as 'class 1' and < 4 as 'class 0'. We incorporate logistic regression model approach for classification and study the impact of L1 & L2 regularisation in various dimensional settings. In addition to on low-dimensional dataset with original 50 features, we create another high-dimensional dataset with around 1275 features with 2-way feature interactions. It is shown that in low-dimensions, regularisation doesn't give much improvement over the standard regression. In high-dimensional data L1 regularisation for feature selection followed by model training using L2 regularisation gives the best results as compared to L1 & L2 in isolation.

2 Introduction

Logistic is regression is one of the novel parametric techniques used in case of classification problems. But it is well known that logistic regression overfits in case of high-dimensional data where there are many irrelevant features. L1 & L2 are the two regularisation techniques that induce penalty on the higher weight values and in turn make the model less prone to overfitting. It is also suggested that L1 regularisation performs the best for feature selection by introducing sparse weight matrix where as L2 regularisation works better for predicting using the model. Here we study both the methods on low-dimensional and high-dimensional settings.

3 Method

3.1 Logistic Regression[1]

Logistic Regression is a parametric method of classification used for predicting binary dependent target variables. Probability or Odds of "success" is modeled as a logit transformation of linear combination of the features.
For a data with N records with m dimensions, logistic regression model is defined as:

$$\pi_i = Pr(Y_i = 1|X) = \frac{e^W}{1 + e^W}$$
$$W = \sum_{j=1}^{m}(\beta_j x_j)$$

Logit transformation of π is given as:

$$logit(\pi_i) = log(\frac{\pi_i}{1-\pi_i}) = \sum_{j=1}^{m}(\beta_j x_j)$$

Parameters of the model β are estimated from Maximum Likelihood Estimation method, that is the values of β_j that maximizes:

$$L(X|\beta) = \prod_{i=1}^{N} \pi_i^{y_i}(1-\pi_i)^{n_i - y_i}$$

Gradient descent method is used for MLE where the loss function is defined as:

$$Loss = \sum_{i=1}^{N}(y_i log(p_i) + (1-y_i) log(1-p_i)$$

3.2 Regularisation

Regularisation is a technique to prevent to over-fitting in Machine Learning. A regularisation term is added to the Loss function in order to penalize higher values for weights.
L1 regularisation specifically penalises the absolute value of weights and produces sparse weight matrix in case of high-dimensional data.

$$Loss_{L1} = \sum_{i=1}^{N}(y_i log(p_i) + (1-y_i) log(1-p_i) + \lambda \sum_{j=1}^{M} |w_j|$$

L2 regularisation penalises the squared value of weights and reduces the chances of overfitting in case of high-dimensional noisy data.

$$Loss_{L2} = \sum_{i=1}^{N}(y_i log(p_i) + (1-y_i) log(1-p_i) + \lambda \sum_{j=1}^{M} w_j^2$$

Where λ called as regularisation coefficient is a hyperparamter and is tuned usually through cross validation technique.

3.3 Modeling

It is generally seen that the L1 regularisation generates sparse weight matrix and hence can be a choice for feature selection in case of high-dimensional data where there could be many irrelevant features. L2 regularisation on the other hand is rotationally invariant and provides better model for prediction.

4 Experiments

4.1 Feature Engineering

The Yelp dataset contains 5000 Yelp reviews with word counts of 50 keywords as features and the rating as a binary target feature of 1's and 0's representing a rating of ≥ 4 and < 4 respectively. A dataset of 1000 additional reviews was used for testing the best model.

As a part of feature engineering, in addition to the original features, two-way interaction features were added considering all the possible feature combinations. Two-way interaction of features $x\&y$ is computed as product of features $(x * y)$. Thus the 50 base features in the data lead to 1225 additional combinations of interaction features. Here onwards we refer to dataset with 50 features as low-dimensional data and the one with additional 1225 features (total of 1775 features) as high-dimensional data.

4.2 Model Implementation

Logistic Regression with gradient descent solver was implemented in Python. In addition, L2 regularisation technique was also implemented. Since L1 regularisation doesn't have an analytical solution form, standard library implementation from $scikit - learn$ module was used.

In the project we study the effect of L1 and L2 regularisation techniques in case of low-dimensional and high-dimensional data for different values of regularisation coefficients.

A modeling choice of using L1 regularisation for feature selection and using L2 regularisation for prediction is also studied for the pertinent data and shown to perform better than isolated L1 and L2 regularisation models in case of high-dimensional data.

4.3 Model Validation

Training data of 5000 rows was split into $3-$folds and cross validation technique was used for model validation and parameter tuning. Overall classification accuracy and ROC area under the curve metrics were used to the model performance.

5 Results

5.1 Low-dimensional data with original 50 features

Standard logistic regression was performed without any regularisation on the low-dimensional data, it gives an accuracy of 0.709 and AUC of 0.750.

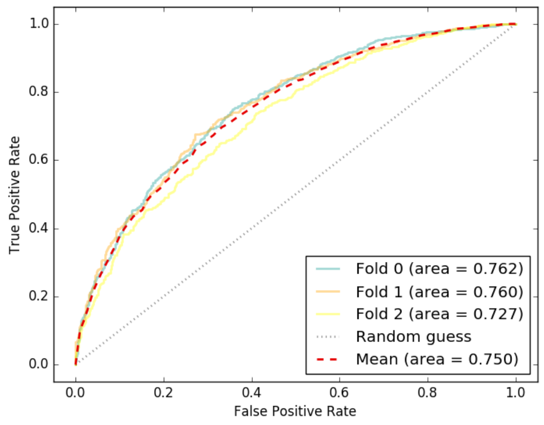
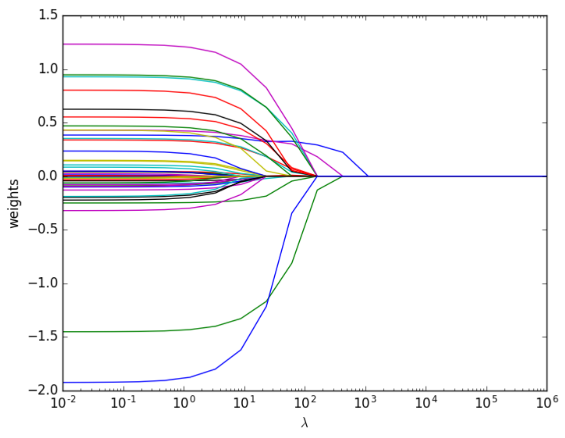
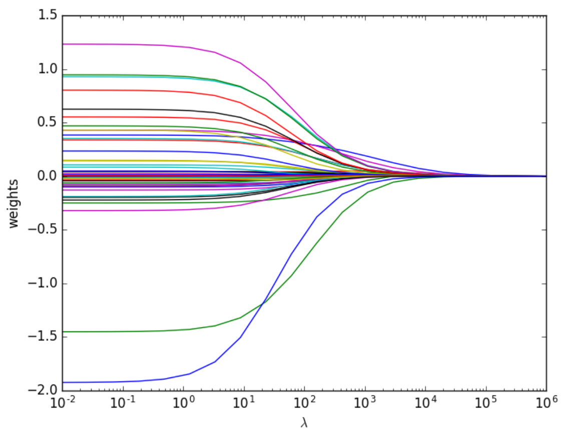


Figure 1: Cross validation ROC for logistic regression without regularisation

Regularisation was added to the above model using both L1 and L2 regularisations for varying values of regularisation coefficients (λ) between $(10^{-2}, 10^{6})$. Figure 2 below compares the charts for L1 and L2 regularisations. It can be seen that the weights for different 50 features for L1 taper off zero earlier than L2 for same λ value.



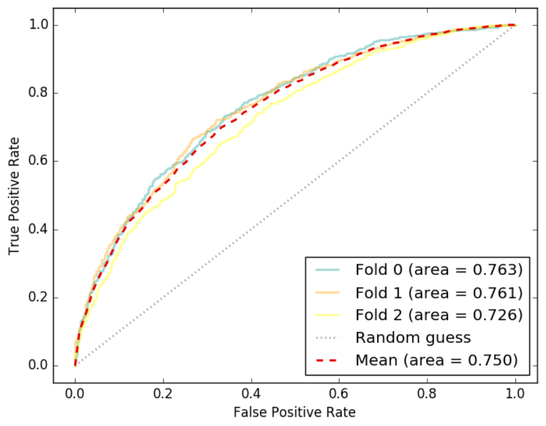
(a) L1 regularisation



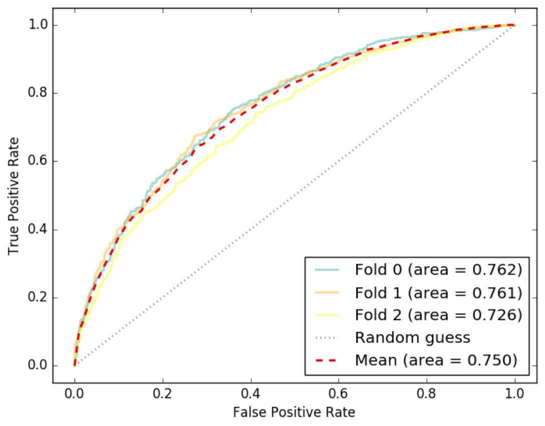
(b) L2 regularisation

Figure 2: Change in weights for varying λ

Figure 3 compares the ROC charts for L1 and L2 regularisations. Best value for λ was selected from cross validation. It can be seen that both the regularisation settings perform similar to the model without any regularisation.



(a) L1 regularisation



(b) L2 regularisation

Figure 3: Cross validation ROC for logistic regression with regularisation

Following table summarises the results for various approaches. It can be seen that all the three models perform almost similarly. It can be seen that in case of low-dimensional data, for the given example there are not many irrelevant features hence regularisation doesn't bear much improvement in the model performance.

Model	Accuracy (+/- 2σ)	AUC (+/- 2σ)
Logistic without regularisation	0.709 (+/- 0.03)	0.750 (+/- 0.03)
Logistic with L1 regularisation	0.710 (+/- 0.02)	0.751 (+/- 0.03)
Logistic with L2 regularisation	0.708 (+/- 0.03)	0.750 (+/- 0.03)

5.2 High-dimensional data with 1277 features

Interaction features were added to the above datasets anticipating an improve in model performance. Firstly, standard logistic regression was performed without any regularisation on the high-dimensional data. Due to presence of irrelevant features, without regularisation, logistic regression overfits and the model performs worse than that on the original 50 features.

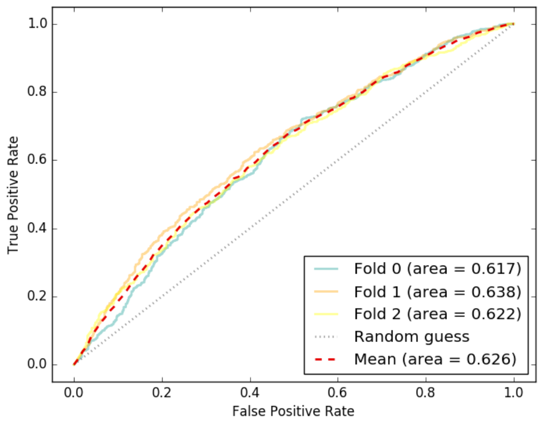
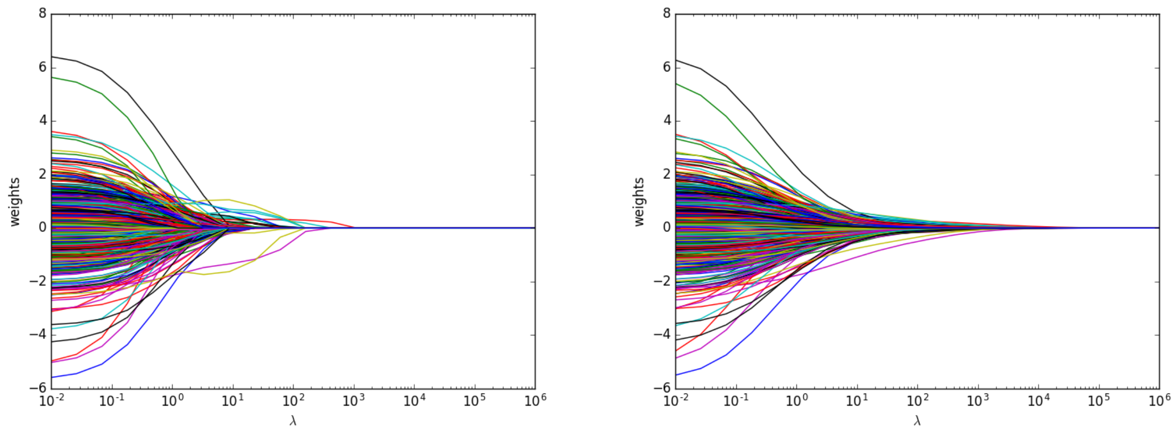


Figure 4: Cross validation ROC for logistic regression without regularisation

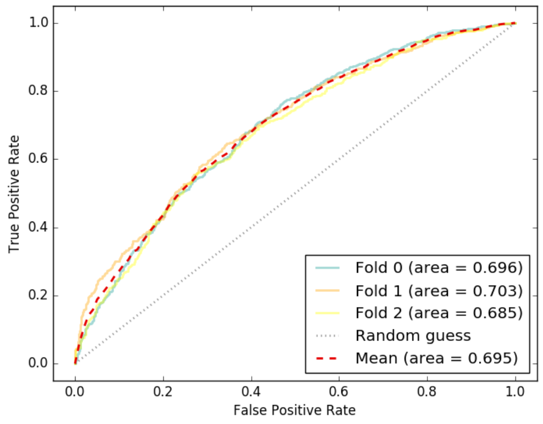
Regularisation was added to the above model using both L1 and L2 regularisations for varying values of regularisation coefficients (λ) between $(10^{-2}, 10^{6})$. Figure 5 below compares the charts for L1 and L2 regularisations. It can be seen that the weights for different 1277 features for L1 generates sparse weights as compared to L2.



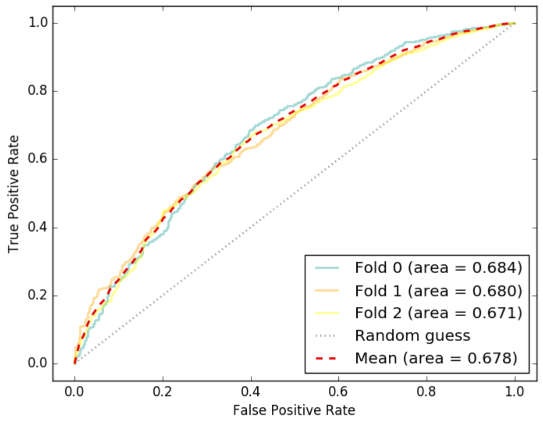
(a) L1 regularisation (b) L2 regularisation

Figure 5: Change in weights for varying λ

Figure 6 compares the ROC charts for L1 and L2 regularisations. Best value for λ was selected from cross validation. It can be seen that both the regularisation settings perform almost similarly. It can also be seen that both these models perform better than the one without regularisation.



(a) L1 regularisation



(b) L2 regularisation

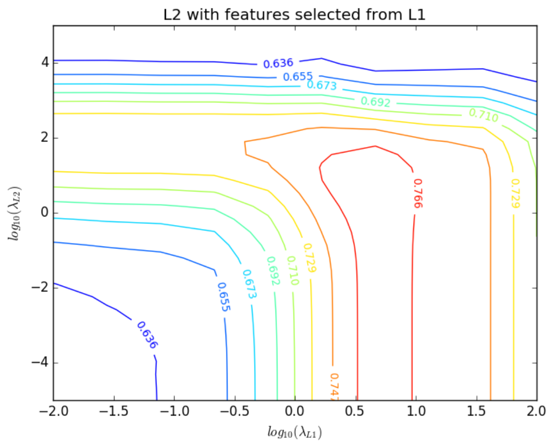
Figure 6: Cross validation ROC for logistic regression with regularisation

But with additional interaction features in high-dimensional data, even after regularisation the prediction performance is not better than the model on original 50 features.

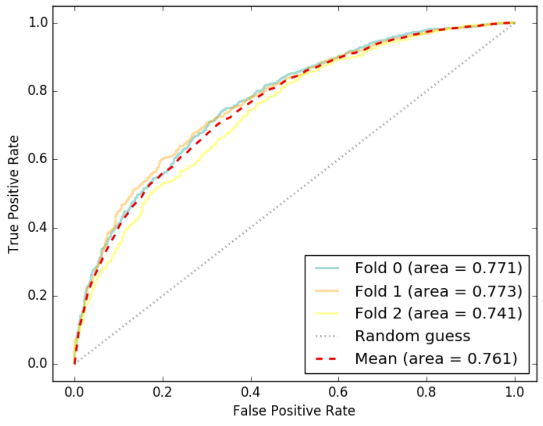
5.3 High-dimensional data with L1 for feature selection and L2 for prediction

Upon considering the notion that L1 produces sparse weight matrix and L2 performs better for prediction, a new approach was followed with performing L1 on the high-dimensional data for feature selection to arrive at non-zero weight features. Thus filtered were then used to build a new L2 regularised model for prediction.

Figure 7 (a) shows the performance of the above model for different values of regularisation parameters of L1 & L2. It can be seen that for a specific range of λ_{L1} the above setting gives best results for a wide range of λ_{L2} values. Figure 7 (b) shows the cross validation ROC for best parameters λ_{L1} & λ_{L2}.



(a) AUC for different λ_{L1} & λ_{L2}



(b) ROC performance

Figure 7: Model with L1 feature selection & L2 prediction

Following table summarises the results for various approaches on high-dimensional data. It can be seen stand alone, L1 & L2 models perform worse than the 50-feature models. But upon using the L1 for feature selection and L2 for prediction model accuracy improves more than rest of the models.

Model	Accuracy (+/- 2σ)	AUC (+/- 2σ)
Logistic without regularisation	0.630 (+/- 0.03)	0.643 (+/- 0.04)
Logistic with L1 regularisation	0.672 (+/- 0.03)	0.698 (+/- 0.03)
Logistic with L2 regularisation	0.657 (+/- 0.03)	0.681 (+/- 0.03)
L2 prediction with L1 regularisation	0.720 (+/- 0.02)	0.764 (+/- 0.04)

5.4 Performance on test data

Running the best performing model on the test data gives an accuracy of 0.7169 and auc score of 0.7488.

6 Conclusion

In low-dimensional data, there are not many irrelevant features and hence regularisation doe not bear much improvement on the model performance. It is shown that logistic regression without any regularisation performs almost same as L1 and L2 regularised models.

In high-dimensional data by adding interaction features introduces many irrelevant features to the data. Logistic regression without any regularisation overfits the data and doesn't perform well. L1 performs better than L2 regularisation as it does inherent feature selection in the model. L2 regularisation performance is lower than that of L1.

It could be a reason that the amount sample data required to learn "well" for L1 regularisation grows *logarithamically* in number of irrelevant features where in case of L2 it grows *linearly* in worst case [2].

Lastly, it is also seen that L2, being a better regulariser as compared L1 for prediction, performs the best on subset of features in high-dimensional data, with features selected using L1 regularisation.

7 Reference

[1] https://onlinecourses.science.psu.edu/stat504/node/150
[2] Feature selection, L1 vs. L2 regularisation, and rotational invariance by Andrew NG.

8 Appendices

```
import numpy as np


class LogisticRegression:

    def __init__(self,alpha=1e-4, niter=1e+4, tolerence=1e-5,
       L2regular = False,lambreg = 0.01):
        '''
        :param alpha: learning rate
        :param niter: number of iterations before convergence
        :param tolerence: the delta loglikelihood for convergence
        :param L2regular: Boolean - for L2 regulairization cost
        :param lambreg: lambda for L2 regularization
        '''

        self.alpha = alpha
        self.niter = niter
        self.tolerence = tolerence
        self.regularization = L2regular
        self.lamb = lambreg

    def train(self,data,target,features = None):
        '''
        :param data: input pandas dataframe
        :param target: the target label
        :param features: python list of explnatory features
        :return: weights for each feature
        '''

        self.data = data.copy()
        self.features = features.copy()
        self.target = target.copy()

        self.x = self.data[self.features]
        self.x['constant'] = 1    # add a constant vector
        self.features.insert(0, "constant")
        self.x = self.x[self.features]

        self.y = np.concatenate(self.data[target].values)

        if self.regularization :
            self.weights = self.GradientDescent()
        else :
            self.weights = self.GradientDescentL2Regular()
        return self.weights


    def sigmoid(self):
        '''
        :return: computes sigmoid for given data vector and
           weights
        '''

        g = 1/(1+np.exp(-self.x.dot(self.weights)))
        return g

    def logLikelihood(self):
        '''
        :return: compute log likelihood cost function
        '''

        prob = self.sigmoid()
        loglikelihood = -1*(np.sum(self.y*np.log(prob+1e-50)+(1-
           self.y)*(1-np.log(1-prob+1e-50))))
        return loglikelihood


    def GradientDescent(self):
        '''
        :return: computes weights gradient descent algorithm
        '''

        # initialize weights to zero
        self.weights = np.array([0 for f in self.features])
        prob = self.sigmoid()
        logllprevious = self.logLikelihood()

        # compute gradient
        gradient = self.x.multiply(self.y - prob, axis=0).sum(
           axis=0)

        # update weights till maximum iterations
        for iteration in np.arange(0,self.niter):
            self.weights = self.weights + (self.alpha * gradient)
            logll = self.logLikelihood()

            # check for convergence
            if np.abs(logll - logllprevious) < self.tolerence:
                break

            # weight update
            logllprevious = logll
            prob = self.sigmoid()
            gradient = self.x.multiply(self.y - prob, axis=0).sum
               (axis=0)

        return self.weights


    def logLikelihoodL2(self):
        '''
        :return: compute log likelihood cost function with L2
           penalization
        '''

        prob = self.sigmoid()
        loglikelihoodL2 = (-1*(np.sum(self.y*np.log(prob+1e-50)
           +(1-self.y)*(1-np.log(1-prob+1e-50))))) + ((self.lamb
           /(2*self.x.shape[0]))*(self.weights.dot(self.weights))
           )
        return loglikelihoodL2
    def GradientDescentL2Regular(self):
        '''
        :return: computes weights gradient descent algorithm with
            L2 regularization
        '''

        # initialize weights to zero
        self.weights = np.array([0 for f in self.features])
        prob = self.sigmoid()
        logllprevious = self.logLikelihoodL2()

        # compute gradient
        gradient = self.x.multiply(self.y - prob, axis=0).sum(
           axis=0)

        # update weights till maximum iterations
        for iteration in np.arange(0,self.niter):
            self.weights = (self.weights*(1-(self.alpha*self.lamb
               /self.x.shape[0]))) + (self.alpha * gradient)
            logll = self.logLikelihoodL2()

            # check for convergence
            if np.abs(logll - logllprevious) < self.tolerence:
                break

            # weight update
            logllprevious = logll
            prob = self.sigmoid()
            gradient = self.x.multiply(self.y - prob, axis=0).sum
               (axis=0)

        return self.weights

    def predict(self, newdata, type="raw", classthreshold=0.5):
        '''
        :param newdata: new data for which the prediction is made
        :param type: return prediction type : raw & class
        :return:
        '''

        ndata = newdata.copy()

        ndata['constant'] = 1  # add a constant vector
        ndata = ndata[self.features]
        prediction = 1 / (1 + np.exp(-ndata.dot(self.weights)))

        # check return type
        if type == "class":
            prediction = (prediction >= classthreshold) * 1

        return prediction
```

Combining Regression and Classification models for ordinal regression problem

Sujith Padaru & Pashupati Hegde

November 25, 2016

1 Abstract

In this report we tackle the problem of predicting the usefulness of a new yelp review based on the bag of words representation of the review. The 'usefulness' is parametrized by vote and hence predicting the vote can be tackled by both regression and classification. Here we take an ensemble approach where we initially consider vote as a continuous parameter and predict it using linear regression and then bin the predicted vote using classification. the intuition for this again comes from the linear regression output. We consider 2 classification models; Bayesian classifier and logistic regression. For logistic regression we define the classification task as rounding up or down given the fractional part of the vote. In Bayesian classifier we measure the degree of belonging of the prediction to every cardinal vote. We observe a combination of linear and logistic regression performing excellently for the given problem.

2 Introduction

Predicting the usefulness of a review by estimating it's vote is an ordinal regression problem as the the prediction should be an integer. The data is noise free and words as features has already been handpicked. In this report we try and compare simple Machine Learning algorithms to solve this problem. A simple classifier likes Naive Bayes classifier is found to be excellent for the text classification. With that as a motivation we try to model the problem as linear regression and learn from the data.

3 Method

3.1 Data

The Yelp dataset contains 5000 Yelp reviews with word counts of 50 keywords as features, and the number of votes the review has gotten.The task is to predict the vote for 1000 additional reviews. the maximum vote that was recorded in training data is 26 and minimum is 0.

3.2 Linear Regression

The vote is assumed to be linear combination of 50 bag of words data. The relation between vote(y) and bag of words(x, w) can be represented as,

$$y = \sum_{i=0}^{50} w_i x_i$$

Here it is assumed that the bag of word data are independent. L1 regularization has been performed as the data is sparse.

3.3 Classification

Ideally the predicted vote from regression for every cardinal vote should have a mean around the exact value with least variance. So rounding the prediction should suffice for a good prediction. However for linear regression it is observed that direct rounding of prediction increases MSE in the training data. So a more sophisticated predictors were devised.

1. **Bayesian Classifier**
   Independent normal Distribution is assumed for each target vote. The parameters, mean and variance of each class is estimated from the linear regression output of the testing data. For every prediction from the linear regression, the class probabilities are calculated. The class which maximises the likelihood is se t as the final prediction.

2. **Logistic Regression**
   Assuming minimal error, for every predicted vote, the choice to take is either to either round up or round down. based on this hypothesis a logistic regression model is built with input as the linear regression prediction and target as a choice, rounding up or down.(Refer Comparing Regularisation Techniques for Logistic Regression)

4 Experiments

4.1 Data

The yelp training data is taken as it is and the features have not been transformed. 70% of the data is randomly chosen as training data and 30% as validation data.

4.2 Linear Regression

Linear Regression with optimizing the Mean square error was implemented in Python. Since L1 regularisation doesn't have an analytical solution form, standard library implementation from $scikit - learn$ module was used. 5-fold cross validation is performed to find the L1 regularization coefficients.

4.3 Classification

Rounding off the predicted fractional votes is modelled in detail. The details of the two approaches used are discussed below.

1. **Bayesian Classifier**
   The training data is divided into 26 clusters, each containing all the reviews which have the same vote. Mean and variance of each cluster is calculated. The individual cluster probabilities are ignored. For every predicted value, the gausiian probability belonging to a particular cluster is calculated using the mean and variance calculated using training data.

2. **Logistic Regression**
   The training data is grouped into 2 categories. One category where the prediction is higher than the actual vote and the the other where it is lower. The following 2 modified datasets are given to logistic regressor and Logistic regressor with L1 factorization

   (a) The fractional part of the prediction from linear regression is given as input and Boolian target(1,0) whether to round up or down.

   (b) The fractional part of the prediction and the training data is given as input and the Boolian target as output.

5 Results

5.1 Linear Regression

Linear Regression out of the shelf gives high accuracy. The Mean square error for linear regression without any regularization is 0.055. The accuracy of the model is apparent in the plot below.

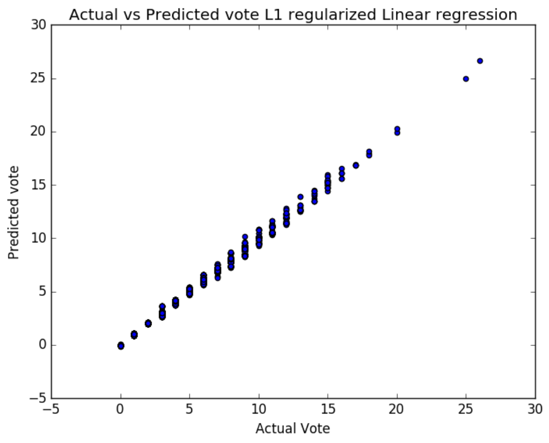


Figure 1: Target Vote vs Prediction, Linear Regression

The Mean Squared Error(MSE) increase if we round off without any criterion.(0.059) A clear pattern in regression is seen which prompts us to develop classification methods on this data.

5.2 Classification

The improvement in the Mean Squared Error when we use linear regression and classifiers is given below. L1 regularization with linear regression didn't add to the accuracy substantially.

Model	Training Data	Validation Data	Test Data
Linear with rounding	0.0591	0.0613	0.064
Linear and Logistic with L1 regularisation	0.0537	0.0566	0.0537
Linear and Bayes Classifier	0.0222	0.0226	0.021
Linear and Bayes Classifier with L1	0.00028	0.00066	0

Table 1: MSE for different ensemble models

Substantial improvement in accuracy is seen when we use Bayes classifier for final prediction. But Logistic regression performs extremely well for this ordinal regression problem. A test error of 0 is observed with all the data and linear regression output as inputs to logistic regression.

6 Conclusion

The experiments prove the potency of simple Machine Learning algorithms when appropriately used. A simple ensemble method is found to be suffice to this problem. It is also inferred that it is best to select a method based on data visualization.

7 References

Feature selection, L1 vs. L2 regularization, and rotational invariance(http://ai.stanford.edu/ ang/papers/icml04-l1l2.pdf) - Andrew Y NG)
Ordinal regression(Wikipedia) https://en.wikipedia.org/wiki/Ordinal$_{r}egression$

8 Appendix

Python Code for the Bayesian Class

```
def bayesclass_predict(Class,model,data):
    x = data
    k = model.predict(x)

    df = DataFrame(index = Class.index.values,columns=x.index.values)
    for i in Class.index.values:
        df.loc[i] = norm.logpdf(x=np.ravel(k), loc=Class.Mean.ix[i], scale=Class.Variance.

    condition = np.ravel([df.max() > -125])
    j =  np.round(np.ravel(k))
    j = j * (~condition)
    j = j + np.ravel(df.idxmax()) * condition

    return j



########### For bayesian classification ###############
Class = DataFrame(columns=['Mean','Variance'],index=np.unique(y[target]))
for i in np.unique(traindf[target]):
    Dummy = y[y['vote'] == i]
    Class['Mean'].loc[i] = np.mean(Dummy.pred_y_lin)
    Class['Variance'].loc[i] = np.var(Dummy.pred_y_lin)
```