Sai Aditya Thalluri
Sharat Tangella

CS 529 Project 2 – Topic Categorization

Introduction

In our project we implemented two classification models: Naïve Bayes and Logistic regression. We are given a training and testing data CSV files. The training data has document id's in the first column, 61188 different word counts for each document and the document label in the last column which belong to one of the 20 different News groups of newsgrouplabels.txt. The vocabulary.txt file consists of 61188 different words. After implementing the two models, we predict the news group labels of the testing data with these two models and get their accuracies. We will give a brief introduction about Naïve Bayes classification and logistic regression before explaining our code.

Naive Bayes Classification

Naive Bayes is one of the most commonly used machine learning classifier for predictive modelling. It is simple but effective. It is a probabilistic classifier and it uses Maximum A Posterior hypothesis for Classification. Naive Bayes Classifiers are a family of classification algorithms based on Bayes Theorem. They all share a common principle which Is every value of a feature is independent of value of other feature.

In a probabilistic classifier, we determine the probability of features X_1, X_2….X_n occurring in each class $Y_1, Y_2 \ldots Y_n$ and return the most likely class. Bayes theorem generally describes the probability of an event by taking the prior knowledge of conditions that are related to the event into account.

Bayes theorem is stated as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

In the project, the different words are taken as features and the newsgroup labels are taken as classes. Now based on the words in the document, we determine which class it belongs to with the help of Naïve Bayes classifier.

From the above equation, we can take A as the class Y and B as the features from X_1 to X_n. Since the denominator term remains constant, we can remove it and modify the above equation and simply state that $P(Y|X_1 \ldots X_n) \propto P(X_1 .. X_n|Y)P(Y)$. Now we make a naïve assumption to the Bayes theorem that all the features are conditional independent. From this we get $P(X_1 .. X_n|Y) = P(X_1|Y) * P(X_2|Y) \ldots . P(X_n|Y)$. The final equation we get is:

$$P(Y|X_1 .. X_n) \propto P(Y) \prod P(X_i|Y)$$

To improve the model, we make additional assumption that $\forall i, j \;\; P(X_i|Y) = P(X_j|Y)$. Thus, we must estimate the parameters to the single distribution $P(X|Y)$ which is equal to $P(X_i|Y)$ using a MAP estimate in addition to estimating $P(X)$ using MLE.

Logistic Regression Classification

Logistic regression is a supervised classification algorithm. Logistic regression is a regression model and it uses sigmoid function to model the data. The sigmoid function is:

$$g(z) = \frac{1}{1 + e^{-z}}$$

Logistic regression uses sigmoid function to transform its output and returns a probability value which can be mapped to two or more classes. There are three types of logistic regression models:

1. Binary
2. Multinomial
3. Ordinal

The goal of logistic regression is to train classifier that can make decision about the new input observation. This is done by learning from a set of training data, a vector of weights and bias term. Each weight is associated with one of the input features and represents how essential the input feature is to the classification decision.

The classifier first multiplies each feature x_i by its weight w_i, sums the weighted features and at last adds the bias term b. The resultant weighted sum is:

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b$$

The value of z is passed to the sigmoid function to create a probability.

We will be needing a loss function that will tell us how close the output given by the classifier is to the actual output. We choose the values of w, b that maximize the log probability of the actual y labels in the training data, given the observations x and this is called as conditional maximum likelihood estimation. The equation of the loss function is:

$$J(\theta) = -\sum_{i=1}^{n}\sum_{k=1}^{K} 1\{y^i = k\}\, logP(y^i = k|x^i;\theta)$$

Where K is the number of total classes and θ is vector that stores coefficients of the class for all words.

To minimize the loss function, we use Gradient Descent. Gradient Descent finds the minimum of a function by finding out the direction in which the slope of the function is rising more steeply in the opposite direction. The loss function in convex for logistic regression. Since a convex function has no local minima as there is only one minimum, the gradient descent is guaranteed to find the minimum starting from any point.

The learning rate ɳ determines the magnitude of amount to move in gradient descent. The change we make is the learning rate times the gradient. It is:

$$w^{t+1} = w^t - \eta \frac{d}{dw} f(x;w)$$

Naive Bayes code explanation:

Initially we load the training and testing data from the csv file and drop the first column from the data which is just the column number. Then we initialize the list for the beta values. We have taken 11 different values for beta from 0.00001 to 1. Then load the label names form the given text file. Now we pass the training and the testing data to function ***classify_testing_documents*** to classify the test data and write the output to a csv. In this function we initially find the length of the vocabulary from the training data by calculating the number of columns and removing 1 from it for the labels. We calculate beta as:

$$beta = \frac{1}{length\ of\ vocabulary}$$

Next, we calculate the likelihoods and priors for the training data.

Pass the training data to the function ***get_likelihoods*** to calculate the likelihoods. This function returns a data frame with the values of the total count of each word in each class and the total number of words in each class. Group the training data by the class label whose size is (number of different classes) x (number of different words) by adding words in the documents which have same class into a single row. Add another column at the to note the total number of words each class s having. Return the resultant data frame to the function ***classify_testing_documents*** as likelihoods.

Pass the training data to the function ***get_priors*.** This function returns the number of total documents that are there for each class. Add a new column to the training data called label_counts and initialize the whole column to 1. Now, group the training data by the class label and add the label_counts columns for the documents having same class label and assign it to a variable document_label_count. We get the total number of documents each class is having. Remove the label_counts column from the training data and return the document_label_count to the function ***classify_testing_documents*** as priors.

By using the likelihoods and priors from the training data, we generate the output labels for the testing data by passing the testing data, priors, likelihoods and beta values to the function ***generate_output_labels*.** Initialize value of alpha as:

$$alpha = 1 + beta$$

Find the total number of documents present from the priors of training data. Calculate the MLE for P(Y) from the formula

$$P(Y_k) = \frac{number\ of\ documents\ labeled\ Y_k}{total\ number\ of\ documents}$$

Get the total_word_counts column from the likelihoods data frame. Calculate the length of vocabulary from the testing data. We convert the testing data and likelihood data-frame to a sparse matrix for better performance. Also, re-calculate the likelihood sparse matrix as per the formula:

$$P(X_i|Y_k) = \frac{(count\ of\ X_i\ in\ Y_k) + (alpha - 1)}{(total\ words\ in\ Y_k) + ((alpha - 1) * (length\ of\ vocab\ list))}$$

Perform a dot product between testing data and likelihoods and add the priors and take class with maximum probability for each document. We classify using the formula:

$$Y^{new} = argmax\left[\log_2\left(P(Y_k)\right) + \sum_i (number\ of\ X_i^{new}) \log_2(P(X_i|Y_k))\right]$$

Return the classified labels for each of the testing document as an array to the function ***classify_testing_documents*** as output labels. Write the output labels to a csv file of desired format.

Finally, plot the accuracies for different values of beta in list using the training data by passing the training data, beta values list and label names to the function ***plot_accuracies.*** Split the training data to 8:2 ratio for train and validation set. Inorder to maintain the same percentage of words in training and validation sets, we are using the ***StratifiedShuffleSplit*** from ***Sklearn***. Calculate the likelihoods and priors of the train data. Get the true output class of documents from the validation set. For each value of beta, Classify the documents from the validation dataset using the likelihoods and priors for the current value of beta. Calculate the confusion matrix using true and predicted outcomes. Now add the accuracies for the current beta value to a list. Plot the graph for the beta values on the X-axis and the corresponding accuracy obtained on the Y-axis.

Logistic Regression code explanation

Initially we load the training and testing data from the csv file and drop the first column from the data which is just the column number. Then take 7 different values of learning rate η and penalty term λ from 0.01 to 0.001. Extract the label names from newsgrouplabels.txt. Pass the training data and the testing data to the function ***classify_testing_documents_lr*** to classify test data and write the output to a CSV. Initialize the learning rate and penalty to 0.05.

Now, pass the training data, label names, learning rate, penalty and number of iterations to the function ***train_weights*** to get the weights tuned on the training data. This function returns a weight matrix by initializing and tuning using training data, label names, learning rate, penalty and number of iterations. Get the length of vocabulary and number of classes and initialize the weights to zero. Create an array X_train using the training data and normalize it.

Pass the training data to the function ***get_delta_matrix*** which returns a sparse matrix of size k x m where k is the number of classes and m is the number of samples in the given dataset. We build a 2d Boolean array of size k x m comparing each of k classes with each instance's label from m instances. Convert the Boolean array to an integer array and return the resultant array as a sparse matrix to function ***train_weights***. Convert the weights and X_train 2d arrays to sparse matrices. In iteration, multiply the weights with X_train and apply the sigmoid function and from the result, calculate the probability sparse matrix by normalizing it. The formula for it is:

$$P(Y|W,X) \sim \exp\ (WX^T)$$

Calculate the change matrix from the delta matrix, probability matrix, X_train, weights and penalty. Now, use the change matrix, weights and learning rate to update the weight using the formula:

$$W^{t+1} = W^t + \eta((\Delta - P(Y|W,X))X - \lambda W^t)$$

Run the iteration for the value of iteration number passed and update the weights. Return these updated weights to ***classify_testing_documents_lr*.**

Pass the weights and testing data to ***generate_output_labels_lr*.** This function generates the labels for the testing data. Get the length of the vocabulary and create X_test from testing data and normalize it. Convert the X_test into a sparse matrix. Multiply the weights with X_test and apply the sigmoid function to get the probabilities. Return the class which has the maximum probability to function ***classify_testing_documents_lr*** and write the predictions to a data frame with the desired format. Write the data frame to a CSV.

We pass the training data, learning rate list, penalty term list and the label names to the function ***plot_accuracies*** where we split the training data in 8:2 ratio randomly for training and validation over different values of learning rate, penalty and calculate their accuracies such that both training and validating data have same equal number of different classes. Create 3 empty lists to store the different values of learning rate, penalty and accuracies for plotting. Iterate through the different values of learning rate and penalty. In the iteration, add the values of learning rate and penalty to the empty lists. Pass the training set, validation set, learning rate value, penalty value, label names and iterations to the ***get_max_accuracy*** function.

The ***get_max_accuracy*** function returns the maximum accuracy for the testing data within the specified number of iterations using training data, learning rate and penalty. Calculate the length of vocabulary, number of classes, initialize the weights to 1. Create X_train array and using training set and normalize it. Get the delta matrix and tune the weights for specified number of iterations. Calculate the updated weights and calculate the accuracies using current weights. If we get the accuracy at-least equal to the previous accuracy, we update the current accuracy or else we terminate the loop. Calculate the confusion matrix at the maximum accuracy and return the maximum accuracy and confusion matrix to ***plot_accuracies*** function.

After we get the maximum accuracy for a particular learning rate and penalty, add the accuracy to the third list. Plot the 3d graph for the values of learning rate, penalty and accuracy.

Conclusion

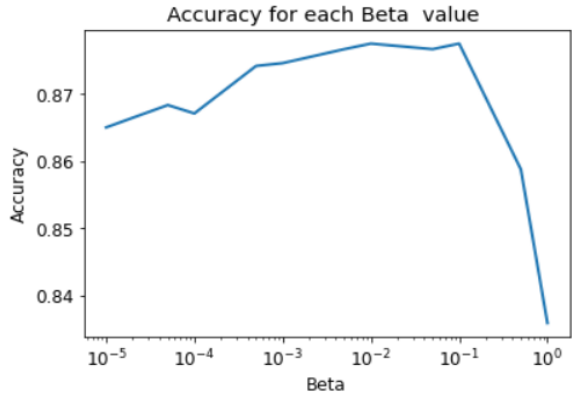
We have classified the dataset based on the class that they belong to with Naïve Bayes and Logistic Regression. We have obtained maximum accuracy with Naïve Bayes classification with the accuracy of 88.6%. The Generative Naïve Bayes performed better than Discriminative Logistic regression.

We are getting more accuracy for Naïve Bayes compared to Logistic Regression. This is because the given dataset is Document Classification and depends on whether a word is present in document or not. This suits more to Conditional Independence of Naïve Bayes and gives more accuracy. Whereas as if we consider datasets such as Sentimental analysis, naïve bayes won’t work properly and logistic is more suitable in such cases.

Answers to questions:

1) It is given that each X_i is sampled from some distribution from its position X_i and the document category Y. We are given vocabulary of 50,000 words and 1000 documents. We have to estimate 50,000 dimensional parameters by using only 1000 samples from it as there are only 1000 words in each document. There won't be sufficient samples in 1000 documents. So it will be difficult to accurately estimate the parameters of this model.

2)

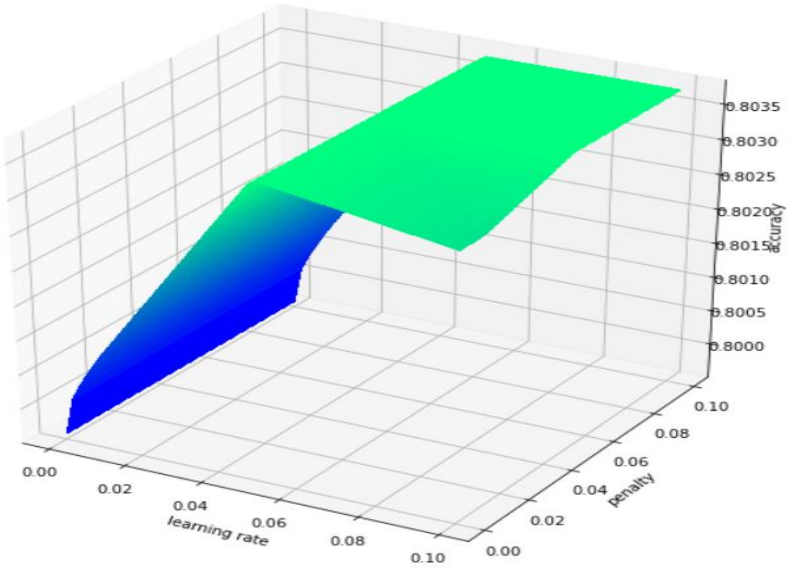


From the plot, it is seen that we obtain the maximum accuracy of 88% with the beta value around 10^{-1}. We get lower accuracy for low value of beta and the accuracy drops with higher beta value.

When the beta value is lower, we get less accuracy because of overfitting because the unique words are given very importance than necessary. When the beta value in higher, the accuracy drops because the unique words are not given importance which leads to underfitting.

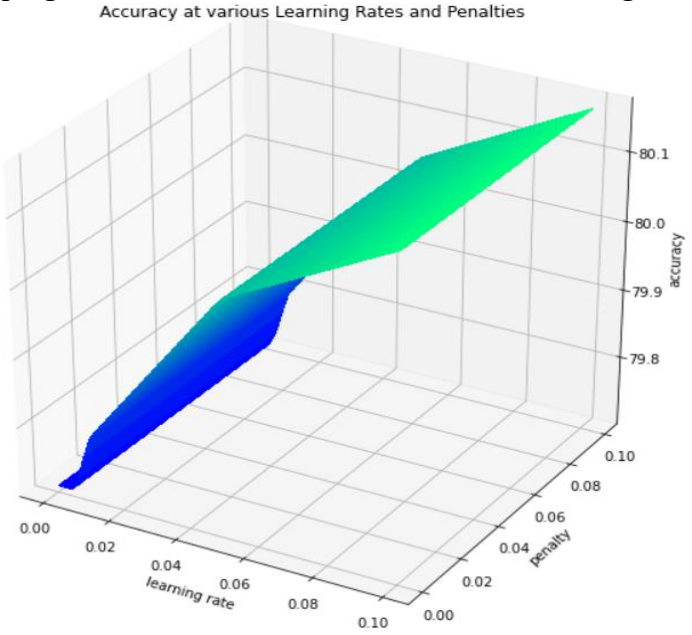
3) Below plots the accuracies of logistic regression classifier at various stopping criteria.

a) Stop calculating further when accuracies start decreasing for each combination of learning rate and penalty.



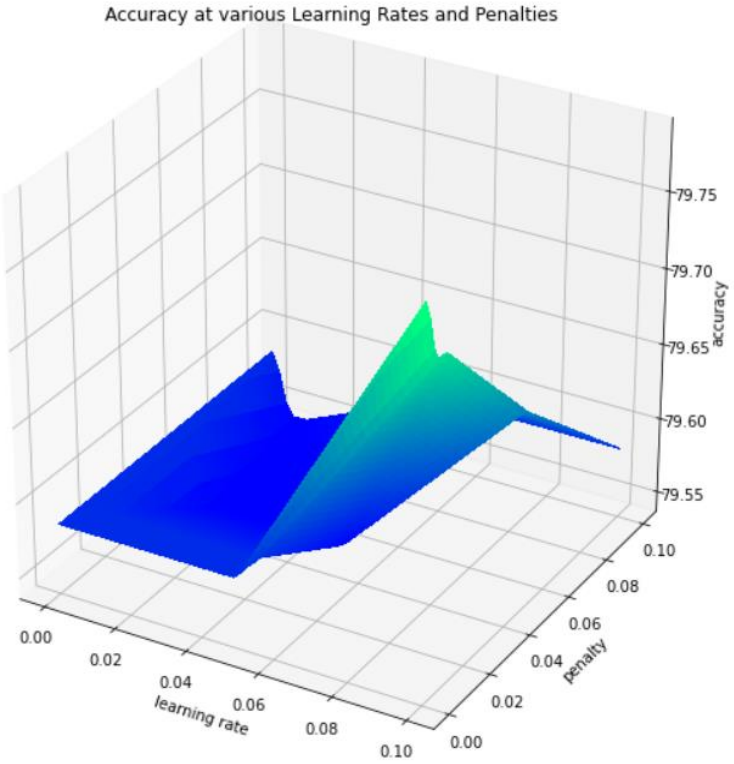
We split the training data in 8:2 ratio for train and validation. We plotted a 3D graph for the accuracy vs learning rate vs penalty. We took the stopping criteria as the difference between the current accuracy and previous accuracy less than 0.00001. The maximum accuracy is obtained at learning rate = 0.097 and penalty = 0.6. The increase in accuracy decreases around at learning rate = 0.045.

b) Stopping criteria of constant 10 iterations for the weights



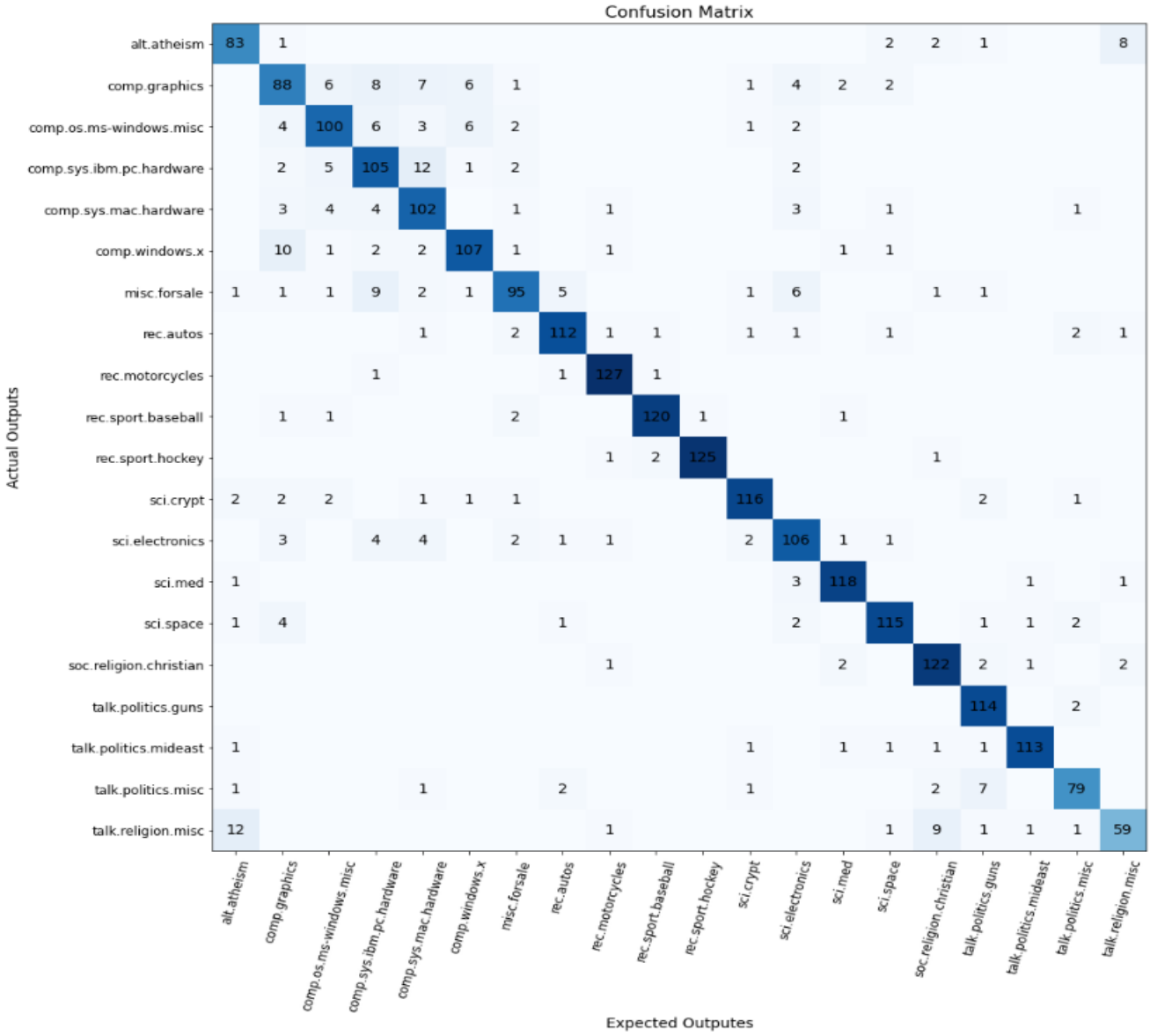
Accuracies don't change much we observe that max accuracy 80.1% at learning rate 0.1 and penalty 0.1.

c) Stopping criteria of constant 100 iterations for the weights.



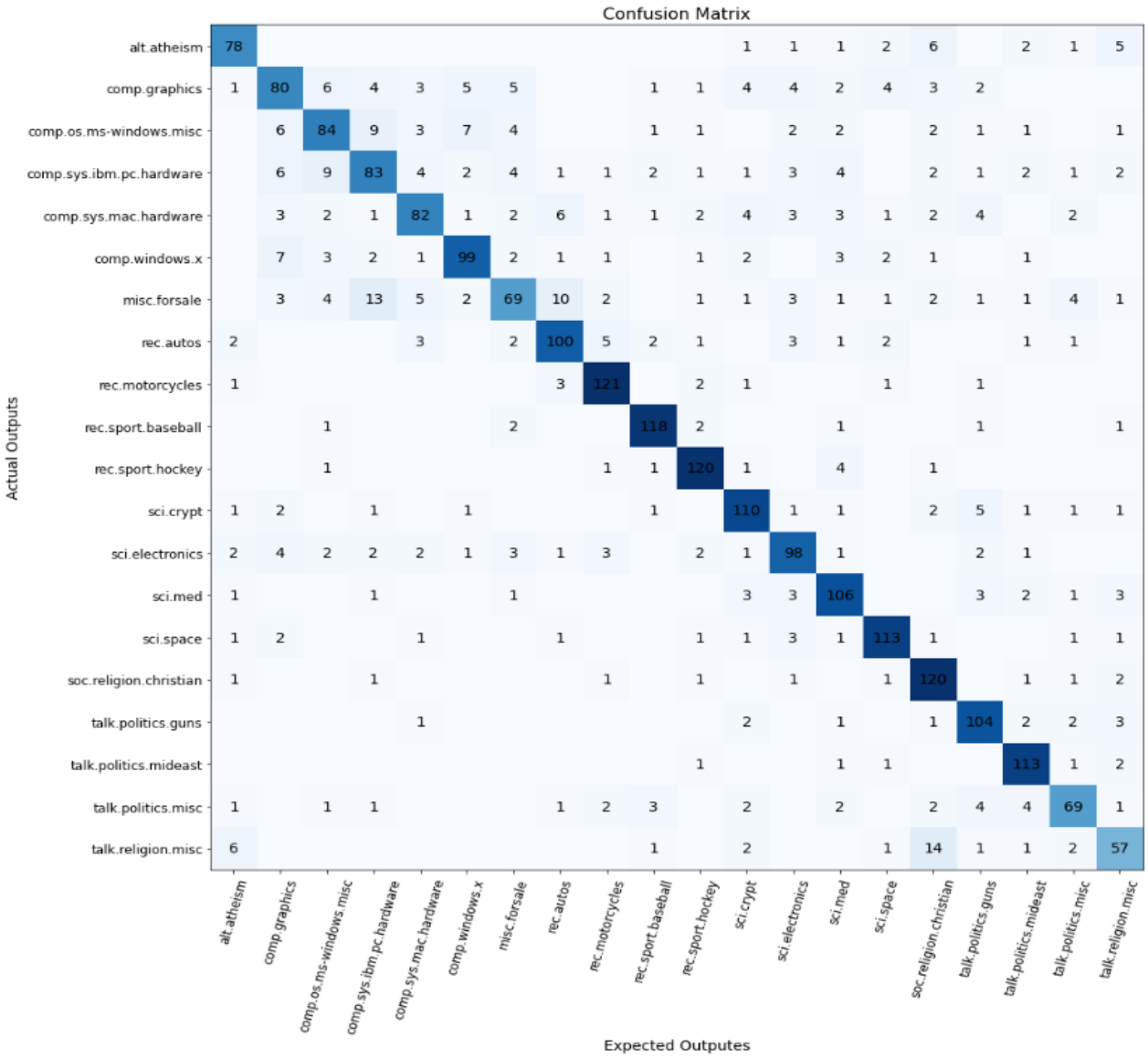
We can see that with increase in the number of iterations as constant for all learning rates and penalties, accuracies don't vary much. This means that all pairs of learning rate and penalty will converge at almost same accuracies with increase in iterations and they won't move further. If we consider 1000 or 10000 iterations, then all the accuracies will be same and it will be a plane graph.

4) The total testing accuracy that we obtained is 88.5% for Naïve Bayes classification.



The above diagram depicts the confusion matrix for the Naïve Bayes classification. We split the training data in 8:2 ratio for training and validation.

The testing accuracy we obtained for Logistic regression is 80.166%. We split the training data in 8:2 ratio for training and validation. The below diagram depicts the confusion matrix for the Logistic regression classification.



5) From the confusion matrix of Naïve Bayes it is seen that sub-topics in the group comp highly get confused with one other and topics with comp and sci.electronics gets confused with each other. This is possibly because these all subtopics have similar words. Also, religious topics such as talk.religion.misc, alt.atheism and soc.religion.christian gets confused with each other because they all have similar words in the documents.

This similar pattern is also observed in Logistic regression. Sub-topics of class comp are confused with one other and misc.forsale is confused with comp.sys.ibm.pc.hardware and rec.autos. This is because they have similar words in the documents of these classes.

6) According to information theory, we can use entropy to rank the words in the given dataset. Entropy defines the amount of information we extract from each word. To perform this, we firstly calculate the MAP likelihood matrix of size kxm where k is number of classes and m is number of instances. Then we convert the probabilities into entropies using the below formula.

$$Entropy = -P(Xi|Y)\log\left(P(Xi|Y)\right)$$

Then we converge the kxm matrix of entropies into 1xm by adding them column size. Sort them for smallest to highest and we take the first 100 features. Features with smallest entropies represent the highest information gain.

7) We implemented the above method for the naïve bayes classifier. It is written as function get_top_100_words in the file NaiveBayes.py. We got the below words.

['lunisolar' 'dict' 'foreach' 'acro' 'vrm' 'dohr' 'cng' 'vafb' 'oldword' 'heatsinked' 'pdqsi' 'electri cy' 'stubbs' 'tethered' 'systeme' 'lepard' 'mesosphere' 'realtech' 'sndplay' 'substr' 'reprocess' ' mreamy' 'dros' 'malouf' 'vramii' 'mdavis' 'rudder' 'yates' 'smilor' 'marcl' 'mohr' 'seqeb' 'aaain nt' 'pinghua' 'seawater' 'ehman' 'convair' 'clee' 'sirtf' 'multimode' 'submillimeter' 'cassegrain' 'montasmm' 'compface' 'xmtextsetstring' 'hardcoding' 'mrminitialize' 'textfield' 'xmtextfiel ds' 'bkilgore' 'xmtexts' 'reparentnotify' 'configurewindow' 'jetskis' 'adfrf' 'bradfrd' 'xymask' 'unwin' 'extravehicular' 'hvidovre' 'kampsax' 'karsten' 'ltx' 'christer' 'hardesty' 'appleus' 'sare x' 'anomalous' 'lyourk' 'yourk' 'oceanic' 'mola' 'voiding' 'brecher' 'magellanic' 'cretaceous' 'g rs' 'ghrs' 'spectroscopic' 'mcontent' 'reamy' 'bistate' 'fredrick' 'labinger' 'barkwell' 'skyscrape rs' 'cfj' 'marcaccio' 'limmitation' 'watchamacalit' 'libxcl' 'faling' 'chrobert' 'veins' 'orpington' 'blanketing' 'everbody' 'hfm' 'qjmf' 'miyamoto']

8) We think that there is some data set bias in the data that we have taken. We do not know in which year this data is taken from. It may be possible that the documents have information that is related to that particular year from which it was taken and that information may not be seen in today's data. For example, some of the documents talk about sports, politics and electronic gadgets. The sports teams, politicians, and the old gadgets may not be referred in the data that we collect today. So, we think that there is some data bias.