**DS5010 Final Project Report**

1. **Title:**

Discriminative and Generative Classifiers – Logistic Regression Classifier and Naïve Bayes Classifier

1. **Authors:**

* Tanmay P Khokle
* Heeru Srichand Ahuja
* Narsing Rao Akula

1. **Github Link:**
2. **Summary:**

This project aims to deliver a tool that can perform classification on a data set using two classifiers- the Naïve Bayes Classifier and the Logistic Regression Classifier. This package mainly targets binomial classification problems.

The Naïve Bayes Classifier is a probabilistic classifier and works on the Bayes Theorem with the assumption of the features being independent of each other (which is why it is considered Naïve). Logistic Regression Classifier utilises a logistic function to get probability estimates. It then learns and updates weights for the training features based on gradient descent.

Without making use of inbuilt or external pre-defined classification tools, the project uses the libraries Numpy and Pandas to enable fast vectorized computation and build the classifiers from scratch.

1. **Design:**

This package is designed with the following components:

* A Module for Logistic Regression Classifier (LogisticRegression.py)

This module has methods with calculate:

* Sigmoid activation (Probability estimate)
* Prediction (Predicted labels)
* Fit (Gradient Descent)
* Cost Function

The sigmoid probability is given by the formula *S(z) = 1/(1+exp(-z))*. It always has an output between 0 and 1, which is great for predicting classes.

The sigmoid value can be used to create a threshold. Any value of probability over 50% would make it likely that the class is 1 or present. If lower, it is categorized as 0 or not present. We can use the features(x) and the trained weights(w) as an input for the sigmoid method. This generates the predicted classes for our classifier.

The cost tells us how far from the actual label the prediction is. It is calculated with the formula *-y\*log(wx)-(1-y)\*log(1-wx)*. The method for gradient descent calculates the gradient for the features with the formula *x(sigmoid(wx) – y)*. The weights are then updated by subtracting the gradient\*learning-rate. This takes the regression towards the minimum.

* A Module for Naïve Bayes Classifier (NaiveBayes.py)

To fill

* Unit testing files for Logistic Regression and Naïve Bayes (TestLR.py, TestNB.py)

Using the python module for *unittest*, The return values and outputs of the important functions of the Classifiers is tested. This includes correct cost calculation, gradient, correct updating of weights, etc.

* A Module demonstrating a use case on a sample dataset for Logistic Regression and Naïve Bayes (ExampleLR.py and exampleNB.py)

These modules use datasets taken from the internet to demonstrate a use case of how one can use the classifiers. The logistic regression example takes the *Breast Cancer Wisconsin* dataset from Kaggle (https://www.kaggle.com/uciml/breast-cancer-wisconsin-data). The Naïve Bayes example takes a sample from *sklearn.datasets,* which comes from sklearn.

* README.md file describing the purpose of the package.
* \_\_init\_\_.py script
* Project report.

1. **Usage:**

The package and modules for the Logistic Regression Classifier and the Naïve Bayes Classifier can be imported into a python module. The classifiers take input of numpy arrays. This is because numpy provides efficient vectorized calculations and we have used it extensively for matrix operations.

After importing the module, one can create an object of the classifier and set the parameters as needed.

For example, an object of the LogisticRegressionClassifier can be created as such:

***algorithm = LogisticRegressionClassifier(learning\_rate = 0.0000001, iterations = 1000)***

We can then call the training method on training labels and predict. The fit method for Logistic Regression also has the option to set verbose to 1, which works similarly to other data science libraries. It will print the loss for every iteration it trains. For example,

***algorithm.fit(x\_train, y\_train)***

***predicted\_y = algorithm.predict(x\_test)***

A similar workflow can be utilized for the Naïve Bayes Classifier. For Logistic Regression, one can also choose the learning rate and iterations for gradient descent.

The files ‘ExampleLR.py’ and ‘exampleNB.py’ include complete examples of using both classifiers on datasets and comparing the accuracy for training iterations.

1. **Discussion:**
2. **Statement of contributions:**
3. **References:**