The Farmers' Protests

A sentiment analysis report

Sundar Raghavan

A quick look at the Farmer's Protests

The 2020–21 Indian farmers' protest is ongoing against the three farm acts passed by the Parliament of India in September 2020. Farmer unions and their representatives have demanded that the laws be repealed. Farm leaders have rejected a Supreme Court of India stay order on the farm laws and the involvement of a Supreme Court-appointed committee. Nine rounds of talks have taken place between the central government and farmers represented by the farm unions between 14 October 2020 and 15 January 2021; all were inconclusive. The acts have been described as "anti-farmer laws" by many farmer unions, and politicians from the opposition also say it would leave farmers at the "mercy of corporates". The farmers have also requested to create an MSP bill to ensure that corporates cannot control prices.

Twitter is a place almost everyone has access to and where people can easily voice out their opinion online for everyone to hear. There are, of course, millions and millions of tweets, and these can be segregated by hashtags and some keywords (this has been illustrated in the scraping process).

Below is a rough sentiment analysis of a day of tweets that people have put out regarding the hot topic of the farmer's protest.

Phase 1: Extraction

There are many ways to extract tweets, but Twitter's API, coupled with the python package 'tweepy', is more than sufficient for some simple sentiment analysis. The tweets extracted have been put out on 27 January 2021. Although it may be a bit naïve to look at only one day's worth of tweets, it can give one an idea about the general mood around the topic, especially when we look at a compact visualisation of the most commonly used words.

Once we have our tweets, we organise them in a pandas data frame. This is done because once we have it in a data frame format, we can do some interesting EDA and further processing using various python tools.

Phase 2: Cleaning

Cleaning text data refers to the process of eliminating all redundant words and characters that will not be of help to our analysis. In the case of tweets, this can include links, hashtags, usernames, RT (retweet), etc. It also includes something known as "stopwords". These words convey no specific emotion/sentiment and are hence not required for our analysis. Examples of stopwords can be I, me, you, she, those, etc. Some stopwords "rt" (retweet) and "th" that remain after cleaning out text like "50th" have also been cleaned out.

Sentiment analysis follows a granular word-by-word approach for which we will have to break all our tweets down into their corresponding list of words and feed that to our final sentiment analysis model.

After passing all our tweets through the preprocess_tweets function, we create a 'Processed Tweet' column where our cleaned text is entered.

Phase 3: Finding sentiment and assigning labels

The textblob library can be used to easily extract the 'polarity' and the 'subjectivity' of a given piece of text. Textblob generates polarity scores ranging from -1 through to 1. -1 being for the text that carries the most "negative" sentiment and 1 for the text that carries the most "positive" sentiment.

We include 'polarity' and 'subjectivity' as columns of our data frame to give our data more meaning.

Now we can run through our new polarity column and assign labels accordingly.

- 1 for positive
- 0 for neutral
- -1 for negative

Ok, these are all the columns we will need in our data frame. We are finally ready to jump into the analysis.

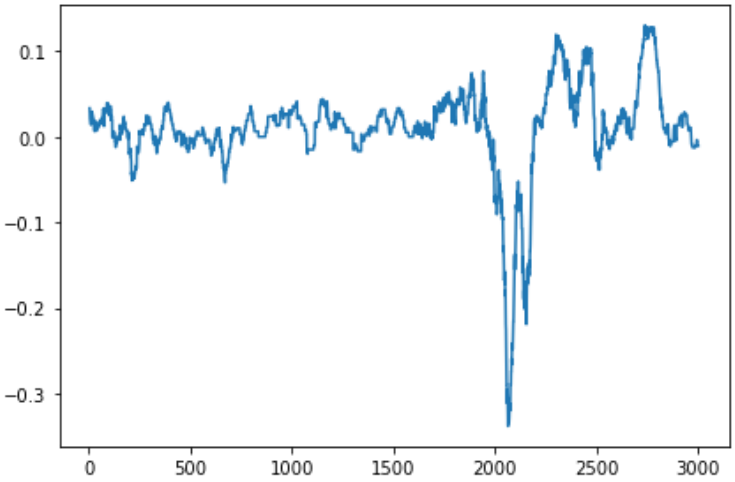
Phase 4: EDA

The simple .describe() pandas command can give us a quick rundown of our data.

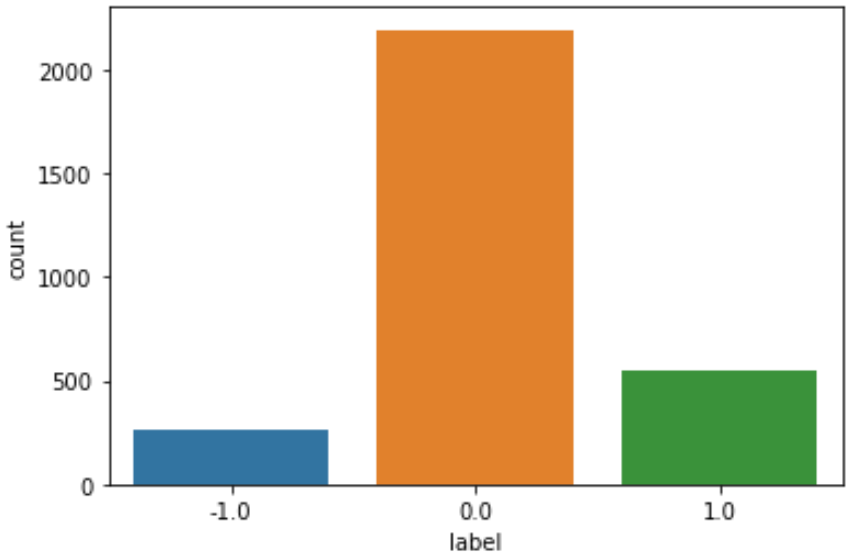
	polarity	subjectivity	label
count	3000.000000	3000.000000	3000.000000
mean	0.009338	0.253869	0.097000
std	0.149111	0.255827	0.509586
min	-1.000000	0.000000	-1.000000
25%	0.000000	0.000000	0.000000
50%	0.000000	0.100000	0.000000
75%	0.000000	0.500000	0.000000
max	0.900000	1.000000	1.000000

The polarity seems to be somewhat positive on an average, and the standard deviation suggests that there aren't many tweets on either of the extremes of our subjectivity score.

We can also plot the polarity to illustrate how the trend has been throughout the day.



A plot of the number of tweets label wise.



A lot of our tweets have a positive sentiment attached to them, a majority have a neutral sentiment and around 10% have a negative sentiment.

Phase 5: Building a classifier

We can build sentiment classifiers using python's sklearn package. This phase involves creating and training a proper ML model to predict the sentiment of a given piece of text and classify it as positive, negative or neutral. 2 classifiers use two different principles, namely Naïve Bayes and Logistic Regression have been created and compared using their confusion matrices.

So, first off, we build a pipeline. This is done so that all the pre-processing, training and predictions can be performed simultaneously. First, we implement the CountVectorizer. This returns a bow (bag of words) model. The bow model converts sentences into a bag of words with no meaning. Every word has a corresponding number assigned to it, which is the count of the number of occurrences. It can be imagined as a vector where each entry is the number of times a word has appeared in the text, and the number of entries will be equal to the number of unique words in the text. This can also be called the term frequency vector or the TF vector for short because it contains the frequency with which each word appears across the text.

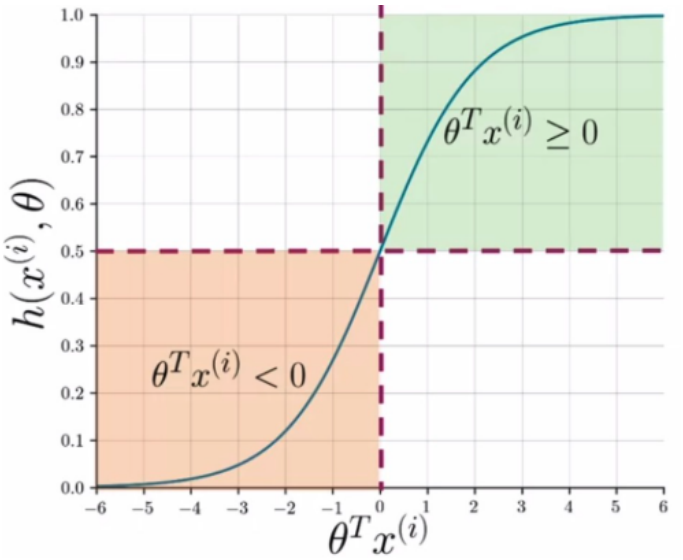
Now what the CountVectorizer will do is

- Clean the data (remove stop words, punctuations, etc)
- Return a sparse matrix that has each word (represented as a unique number) and its document frequency

The second part of our pipeline is the TF-IDF Transformer. This will transform all the integer counts that the words have to their corresponding inverse document frequency (IDF). This will generate an IDF vector. The product of corresponding terms of the TF and the IDF vector will finally give us a vector that has the TF-IDF scores of all words in our text. TF-IDF of a word = TF of the word * IDF of the word. The more common the word across documents, the lower its score and the more unique a word is, the higher the score.

Now that we finally have the tf-idf scores of all our words, we are all set to implement our classifier. As mentioned earlier, two classifiers have been used.

- Naïve Bayes
  - Follows the Naïve Bayes probability principle
  - Uses Bayes theorem for each word in a given piece of text and multiples all the probabilities (independent events)
    - Let's say our sentence is "Our farmers have shown great resilience."
    - The cleaned sentence will look like this ['farmer', 'shown', 'great', 'resilience']
    - Now we want our classifier to tell us whether this sentence carries a positive, negative or neutral sentiment.
    - Naïve Bayes will do the following
      - P(farmer|positive) * P(shown|positive) * P(great|positive) * P(resilience|positive)
      - The above calculation will give us the probability of our sentence carrying a positive sentiment.
      - The same can be done for calculating negative and neutral sentiments as well.

- Logistic Regression
  - Like linear regression, instead of fitting a line to our data points, we fit a sigmoid curve with the z parameter equal to the dot product of the weights and the features (TF-IDF) vector.
  - The value returned by the sigmoid function will help us classify the sentiment as positive or negative based on whether it is greater than or less than 0.5, respectively.
  - 

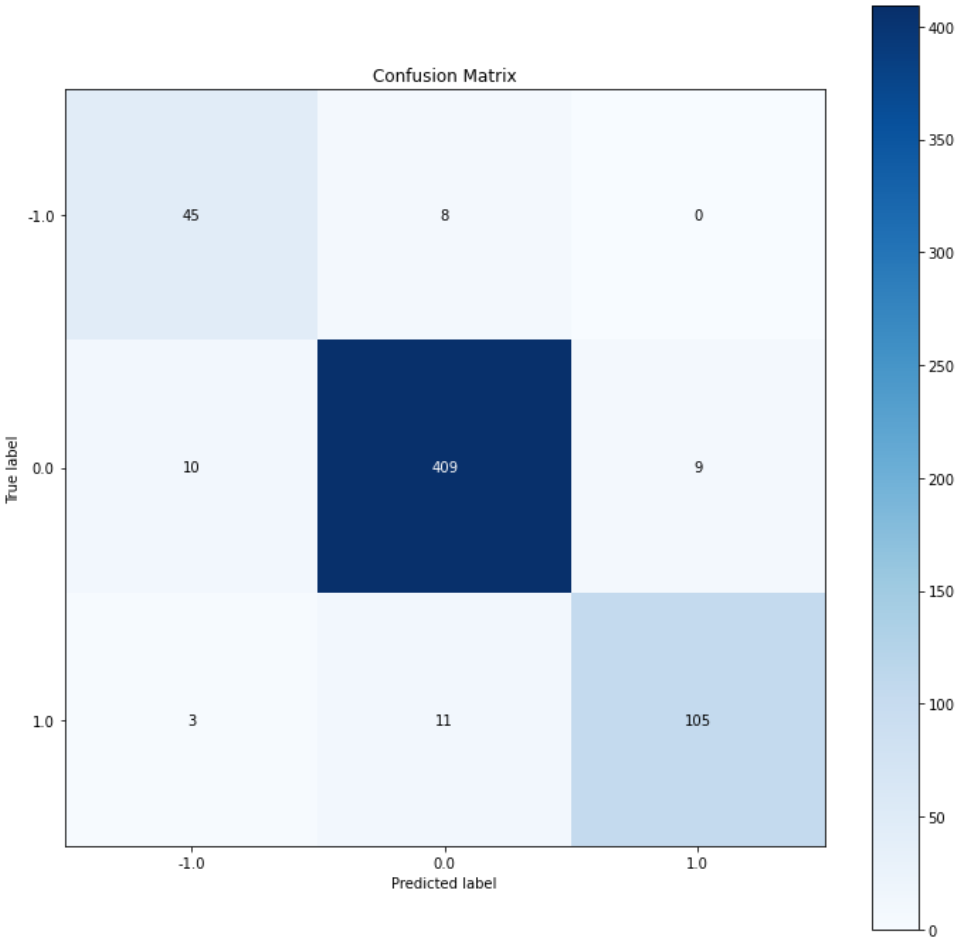
  - As our model gets trained, the weights vector will be iteratively updated till the training process is over.

Finally, we can test both classifiers using our test data and plot the confusion matrices produced by both classifiers and look at their accuracy scores to determine which one works the best for our dataset.

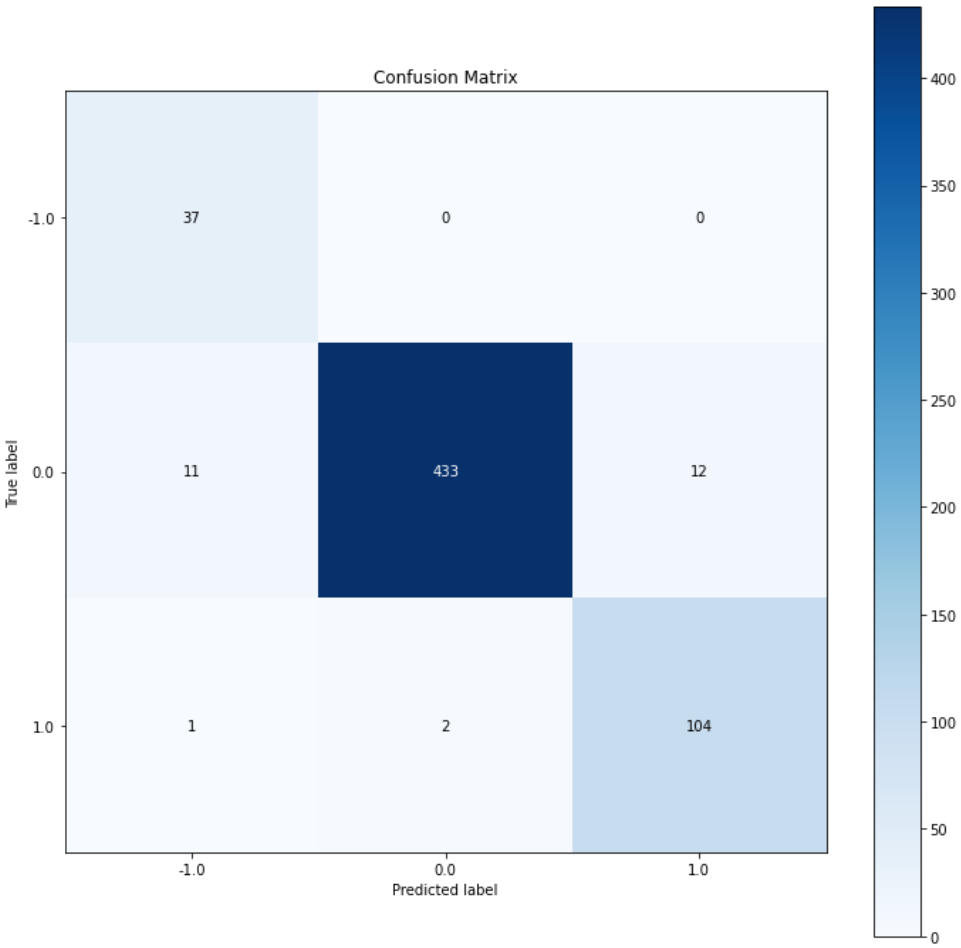
We can see that the confusion matrices given below aren't all that different and the exact accuracy score for the Naïve Bayes classifier was found to be = 93.17 % while that of the logistic regression classifier was found to be = 95.67 %.

So it can be said that a logistic regression-based classification approach has worked a bit better for our dataset.

Naïve Bayes

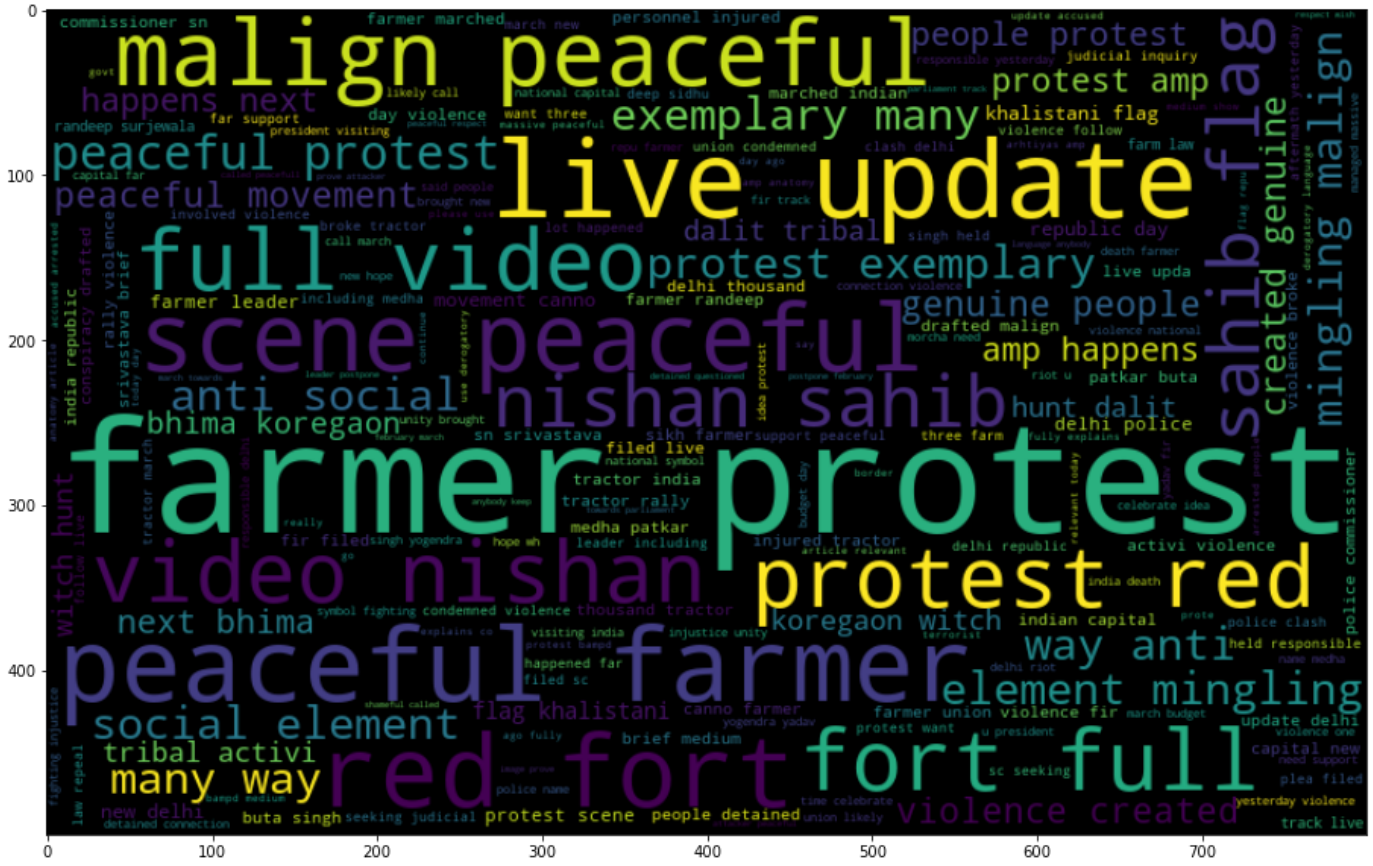


Logistic regression

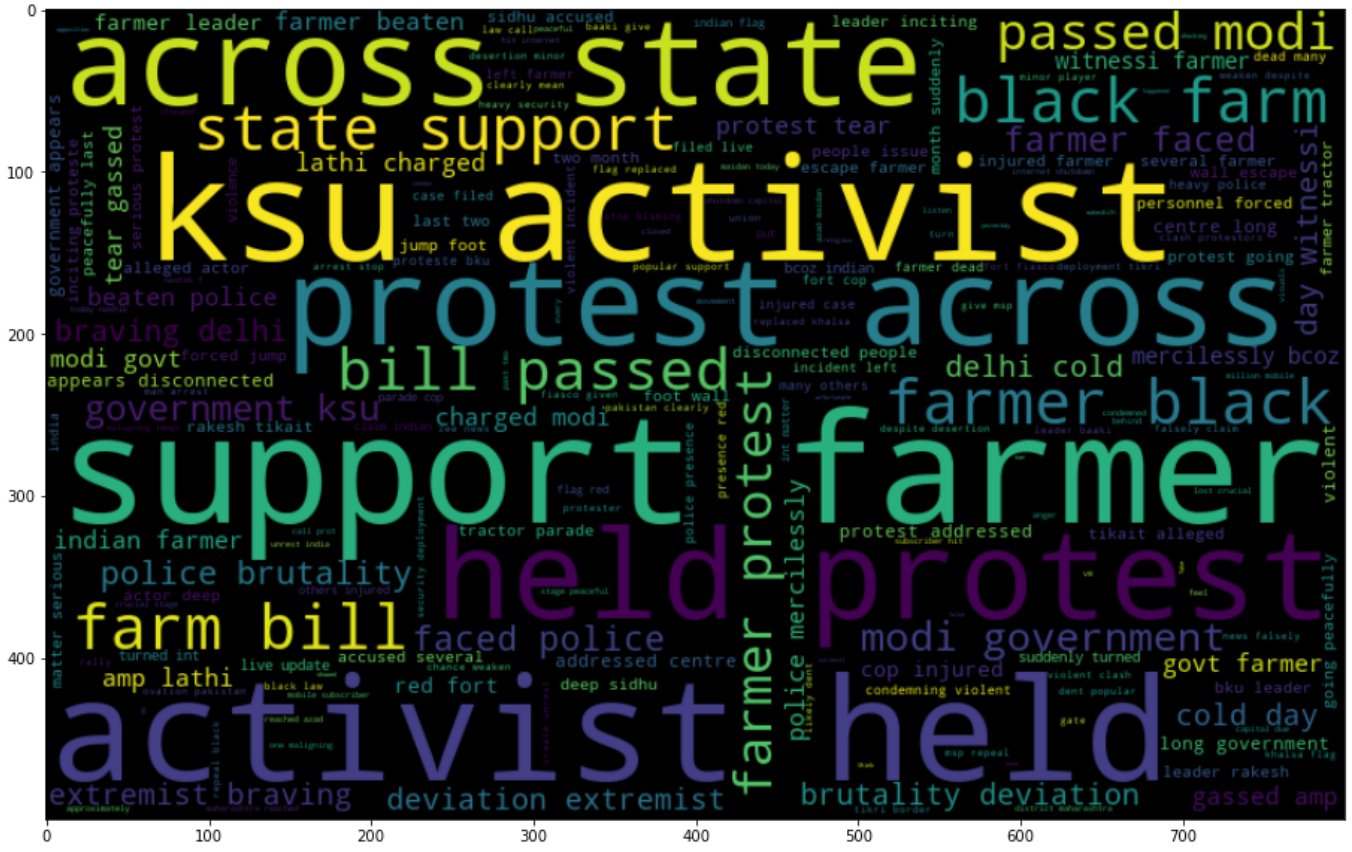


Phase 6: Visualising the Most Used Words

For positive tweets



For negative tweets



Conclusion and limitations

Though I was not expecting to see a more significant number of tweets belonging to the positive sentiment bracket, it became evident why this was the case once I generated the word cloud. Most of the positive tweets had a common backdrop of motivation and support behind them, while the negative ones were filled with criticism targeted at the government and the police. Twitter seems to be with the farmers on this issue, with a severe urgency developing by the day. Moreover, it must be noted that this study does not come without limitations as there are millions of tweets by Indians out there that are in languages other than English (Hindi being the most predominant one). From a statistical viewpoint, there is no guarantee that the sample of tweets scraped for this study had no bias at all as all of them belong to a particular time frame of one particular day. A large number of tweets can be scraped and sampling methods such as Simple Random Sampling can be performed to reduce the bias.

Coming to the models, it was not a surprise observing the Logistic Regression model outperforming the Naïve Bayes model. The former is more sophisticated and rugged mathematically. One argument in support of the above is that Naïve Bayes assumes that all features are conditionally independent, yielding inaccurate results when some of the feature variables are correlated. On the other hand, logistic regression splits the feature space linearly and works well even when variables are correlated.