MULTINOMINAL NAIVE BAYES

Why it is used

Multinominal naïve bayes are used for text classification problems they use BOVW approach. Since the machine can't calculate a probability using strings, the data will require some preprocessing.

The functions in ***myMN_NB*** class:

Training Phase

Fitting the model

The ***fit*** function takes in two parameters a training table of data and its associated training labels. It computes a prior probability for a particular label ***L*** to occur in the training set the result of this computation is passed to the ***vpa*** Matlab function to keep high precision in floating points. It also creates a unique feature dictionary (unique words in training corpus) by calling ***transform_text*** function. Finally, the fit function creates a matrix model by calling the ***populate_feature_matrix.*** The resulting matrix will be of 1 x T, *where T* is the number of unique words or features in our training corpus.

Transforming the text

The ***transform_text*** function allows us to reduce the features in the training corpus by remove noise (useless features) like, punctuations, URLs, Tags, stop words (words not considered necessary for the prediction phase), etc. It also stems the words tokens, by reducing all words in the training corpus to a base form using the built in ***normalizeWords*** MATLAB function. Once the input document has been thoroughly cleaned, we can now reduce the features set further by collapsing all repeating words (features) onto themselves as well as returning the frequency of occurrence of said features. The resulting matrix of features is then reshaped into a 1 x T matrix instead of the T x 1 returned by the MATLAB ***unique*** function.

Populating the feature matrix

The ***populate_feature_matrix*** function create a mathematical representation of the words. It creates a matrix of 1 x T, *where T* is the number of unique features in our training corpus. It then indexes, for every document (example string in training set) and for every in the present document, each feature frequency is mapped to its respective column.

Classification Phase

Predicting a label

Predict

The ***predict*** function predicts the labels for every testing data point. On each iteration the data point (document/test example) is pre-processed by use of the ***transform_text*** helper function. The result matrix of 1 x T unique features in testing example is mapped onto the full words dictionary matrix of 1 x unique feature count in training corpus. Only matching features are retained and a dummy matrix with created for current data point mapping the frequencies of the knowns features to their respective dummy matrix columns. This matrix – a mathematical representation of the word in the current testing point – is then parsed onto the ***predict_one*** function along with the model.

Predict_one

The **predict_one** function, predicts the label for a single test point. It creates a stand in matrix for all training examples associated with the class being investigated. It then computes the likelihood of each feature (word) to occur and multiplies it by the prior probability of that class. This process is repeated for all *T* labels and the winning value is retained as most likely outcome.

Likelihood

The ***likelihood*** function computes the likelihood of a specific word feature to occur in a given class. Doing so in multinominal Naïve Bayes raises an issue. Given a test point *"I have a car and second car as well,"* two labels *"Car"* and *"bicycle"*. Even though the test does show the token 'Car' twice, assuming our training corpus has seen the word 'have' but never in association with 'Car' this likelihood will result in a zero. This is not good as it will introduce high inaccuracies and there is no way to train our model with all possible combinations. It will be overfitted!!

To circumvent this issue "Laplace smoothing" will be applied. For each likelihood, Laplace smoothing will add *alpha = 1* to the frequency of the current feature in the training set. The result will then be divided by the *sum* of the total number of unique feature observations for the current class and the cardinality of the cardinality of the corpus (number of unique features in training set) multiplied by alpha.

The likelihood will also skip any features that didn't occur in our test set. Keeping the word *"Elephant"* as a relevant likelihood feature, when Elephant did not occur in the sentence *"I have a pet name Bagheera the tiger"* would only lead to error in probability.

Laplace smoothing or add 1 smoothing

Will smooth out our probability as resize them according to the context as explained above. It uses the MATLAB ***vpa*** function to return a high precision floating point, since the data employed

is high dimensional it is important to retain as much detail about the likelihood as possible to make a better-informed estimate decision.

Final notes and observations on Multinominal naïve bayes

Although it is great to generalize an algorithm. Personal experimentation has shown that Multinominal naïve bayes (MN) text cleaning phase is not a one size of fits all deal. Remove punctuations and numbers characters could be a good idea in one case, whereas is would drastically reduce performance in another. For example, in a creating a spam catcher model it would be good practice to leave special symbols and numeric chars in a spam sentence of the type “Free money 100000$$$$ call today…” or “you have to pay $$3000 million & to your council.” Whereas in a sentence of the type “I went to japan to and bought a 10000$ car!!” removing these characters would be a great idea.

Comparison between two data set targeting different things

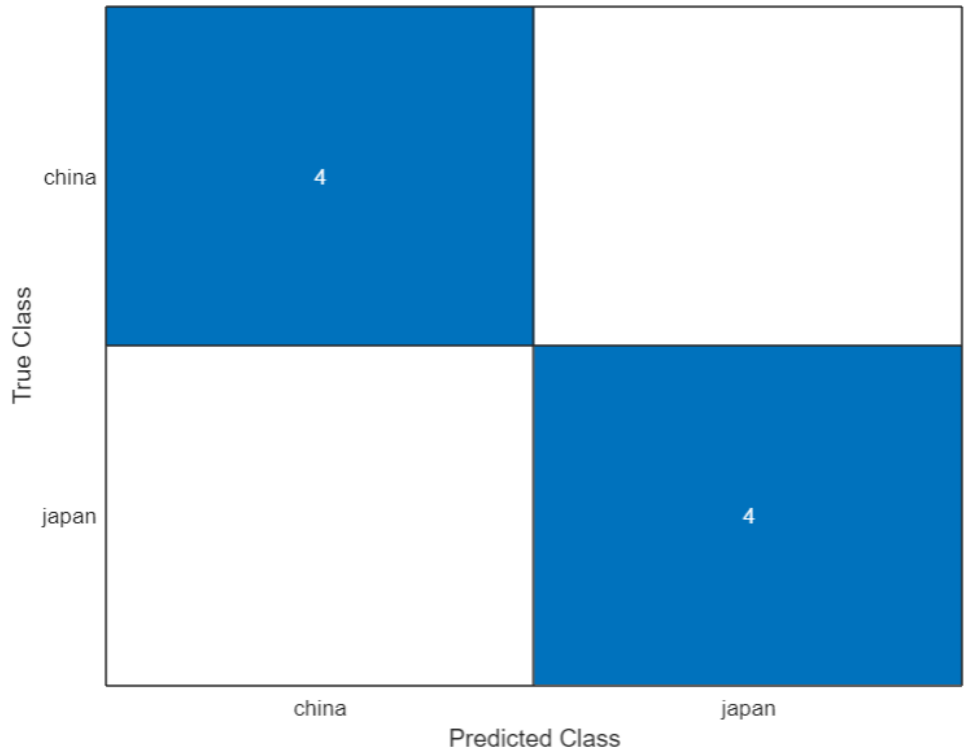

Fig 1: model trained on 32 samples. Simple case of guessing target context in the message algorithm succeeded 100% of the time. Removing punctuations and other unimportant clutter was the right idea.

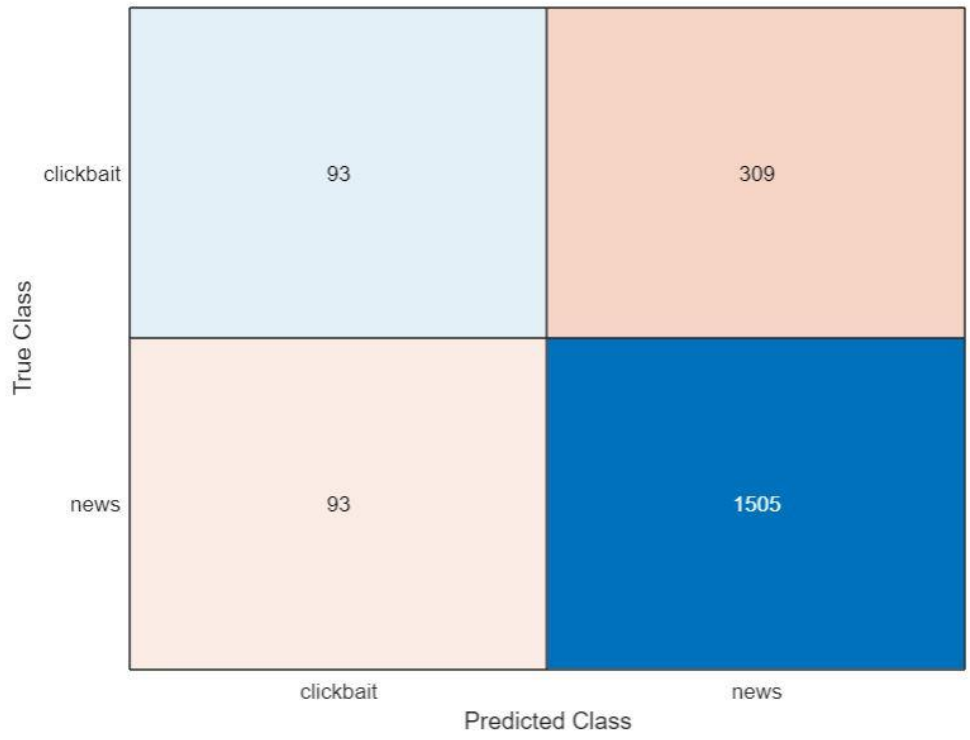

Fig2: Above a classifier trained on 3000 data points to detect clickbait. Shows some of our input's sanitation techniques need further refinement. Difficult to generalize one algorithm for all possible model (good Turing machine may help with this, it looks at combinations of words, rather than smoothing independent features like Laplace smoothing).