CIS 520, Machine Learning, Fall 2017: Project Report
Multi-class Classification of Tweets

Team: Saurav Bose, Anant Maheshwari, Sharadha Srinavasan

1 Introduction & Data

This project aims to do a multi-class classification on tweets. The "success" of any of the methods we have run, is based on the cost returned to us by the leaderboard. The data provided was as follows:

- train.mat : This contained *X_train_bag*, *Y_train* and *train_raw*. As the name suggests, *X_train_bag* was the training data set. This was a sparse matrix, which had under each feature(columns), the number of times each word from the vocabulary set had been seen in the respective tweet (rows). *train_raw* had the raw tweets.
- validation.mat : This contained *X_validation_bag*,and *validation_raw*. These data sets are to validate our classification.
- vocabulary.mat : This contained just a single row matrix of strings containing all possible words in the vocabulary of the data set.

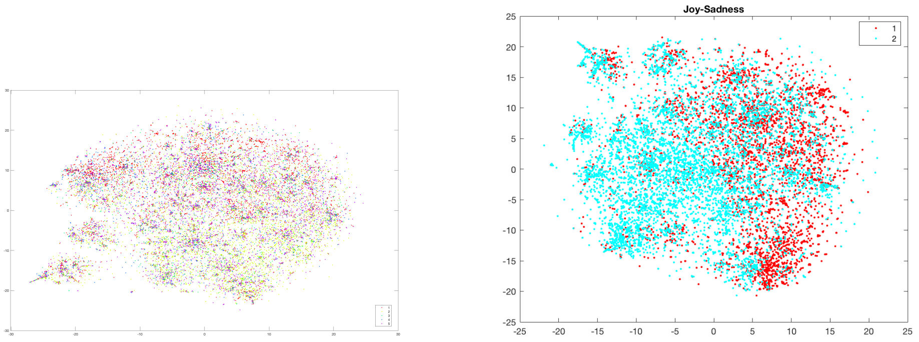

Figure 1: 2D reduction of entire data (t-SNE), Selected Labels: Joy and Sadness

The figure on the left is the 2D representation of the entire dataset using the t-SNE algorithm. No clear patterns were observed so another plot of only contrasting labels (Joy and Sadness) was created as shown on the right. There is a visible division between the joy labels(red) and sadness labels(blue) that could be classified fairly well using a linear decision boundary. This was the first clue that models generating linear decision boundaries can work well with this dataset.

2 Initial Methods - Baseline 1

For this baseline, it was our first attempt to just run a variety of models on the data, without any pre-processing. The methods we tried were as follows:

- **Logistic Regression** We used liblinear package to train the model using multi-class logistic regression. We obtain the probabilities of each observation belonging to class 1 through 5. Here, we used the asymmetric cost matrix multiplied with the obtained probabilities. This gives us the cost of misclassification given a prediction. We chose the label with minimum cost as the prediction.

- **Naive Bayes** We used the 'fitcnb' function in MATLAB with a multinomial distribution to train our model parameters using the given training bag-of-words. This gave us a validation cost of **0.9447**. Since Naive Bayes works well for most text classification purposes, this was our first attempt. Although the obtained cost is pretty decent for a basic model like Naive Bayes, it wasn't the best performing model because of the conditional independence assumption it makes on the features.Since we dealt with a bag of words model, Naive Bayes assumed each word to be independent of the other in a tweet. This is not a great assumption since it misses out on capturing context. An improvement to Naive Bayes could have been achieved by generating bi-grams instead of the currently used uni-grams. Although we did generate bi-grams(See Appendix) from the raw texts to test this hypothesis, the number of features exploded to 134,000 from the current 10,00 and it became too large to model in the specified time. We would like to develop a feature selection routine for the bi-gram model to test the algorithm's performance in the future.

- **SVM** We used the 'svmtrain' function from MATLAB's libsvm package to train our SVM models. The 'svmpredict' function was used to generate probabilities for each label for each tweet. These probabilities were multiplied with the cost matrix to generate mean loss for each label. The label with the lowest mean loss was selected as the final prediction for an input tweet. We trained SVM models with both polynomial and rbf kernels. Cross validation was performed to tune the degree and cost parameters for the polynomial kernel and kernel width and cost parameters for the rbf kernel.

3 Continuing Methods - Baseline 2 and onwards

3.1 Pre-processing the bag of words

After successfully crossing Baseline 1, we started trying many different preprocessing approaches to reduce the number of dimensions and normalize the data in order to get to a lower cost. They are as follows:

- Feature Selection:
  - *Grouping Words which mean the same thing:* For this, we grouped features which meant the exact same word, since we assumed they would imply the same sentiment. For example, we grouped "tomorrow" and "2morrow"
  - *Grouping singular and plural of the same word:* For this, we grouped the singular and plural of nouns such as "text and "texts"
  - *Grouping smileys:* Here, smiley icons which meant the same thing were grouped - such as ":)" and "=]"

- *Removing Numbers:* We assumed that numerical values wouldn't add or take much from the sentiment of a tweet, and for this reason we experimented with removing all number-features. There were about 240 such features.

- *Removing "<user>":* All usernames were represented as <user>. We removed these features as they do not add to the sentiment of the sentence.

- **Term Frequency-Inverse Document Frequency (TF-IDF):** TF-IDF pre-processes to data to remove bias on sentence length and by giving lesser weight to stopwords. We tried two variations of TF-IDF and achieved significantly different results. As a step prior to TF-IDF, we normalized each word count by dividing it by total number of words in a given tweet. These are mentioned below:

  - Regular Method:
$$tfidf(t, D) = f_{t,D} * \log\left(\frac{N}{|d \in D : t \in d|}\right)$$
  - Logarithmic Method:
$$tfidf(t, D) = \log(1 + f_{t,D}) * \log\left(\frac{N}{1 + |d \in D : t \in d|}\right)$$

  We obtained our **best cost** using logarithmic TF-IDF.

- **Term Frequency-Inverse Gravity Moment (TF-IGM):** We read through research papers and found another pre-proceessing approach that supposedly works better than TF-IDF. This methods measure the class-distinguishing power of a term. However, We obtained a cost higher than TF-IDF, so we chose to not use TF-IGM.

3.2 Other models

- **K-Nearest Neighbours**
  Given the nature of the algorithm, testing the model takes a long time as there are 10000 features for each of the 18,000 examples and euclidean distance computation is expensive. Cross-validation cost obtained was not up to the mark (over 1.2). This makes sense because predictions based on euclidean distance similarity in such high dimensions is probably just modeling noise.

- **Principal Component Analysis**
  We tried reducing the dimensionality of the input data using sparse SVD (Singular Value Decomposition) and then picking K eigenvectors with maximum eigenvalues to obtain reduced dimensions. We found two flaws with using this approach, which are as mentioned below:

  - Singular Value Decomposition over a 10000 feature bag of words is very expensive. Since Dimensionality Reduction has to be performed over both training and testing data, it was infeasible for us to utilize PCA for the leaderboard submission.
  - We feel that Principle Component Analysis is not suitable for text classification purposes and it makes the data very complex as a lot of features are removed (sparsity doesn't stay anymore)

- **Linear Discriminant Analysis**
  We used the 'fitcdiscr' function in MATLAB to fit a linear discriminant model to our data. The model assumes X has a Gaussian mixture distribution. The model generated was 1.4GB in size, so we could not test it on the validation set.

4 Results

- **Logistic Regression**

  Regularization Constant = 1 (obtained using cross validation; figure below)

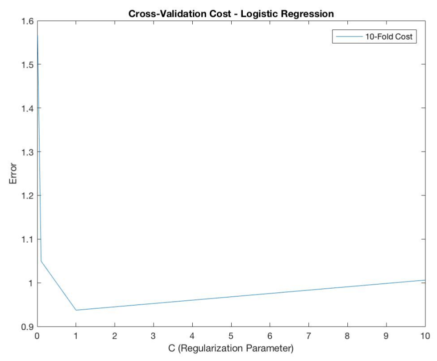

Figure 2: Cross-Validation Error vs Regularization Parameter

  We used the processed data using logarithmic approach of TF-IDF mentioned in the previous section for this model.

  Cost on validation set: 0.9161
  Logistic Regression offered the best performance. Logistic Regression outperformed SVM in our analysis. One hypothesis for this is that SVM optimizes the hinge loss, so it doesn't care how far away from the decision boundary a point is as long as it's beyond the margin. In high dimensional space this can be problematic as decision boundaries are not so clearly defined. In contrast the Logistic Loss is continuous and hence the loss is never zero; it always pushes to reduce misclassification. Moreover, we justified why Naive Bayes couldn't perform as well as Logistic Regression in Section 2.

- **Support Vector Machines** The first attempt was to train an SVM model with polynomial kernel.

  In order to tune the two parameters - degree of polynomial and regularization parameter, a two-dimensional 5-fold cross validation was performed. Polynomial degree of 1 and regularization parameter C = 1 corresponded to the smallest cross-validation errors. In order to visualize these results, we held the regularization parameter C fixed at 1 and plotted the variation of Degree. Thereafter, the degree was fixed at 1 and the regularization parameter C was plotted(Figure 3).
  Cost on validation set: **0.949**

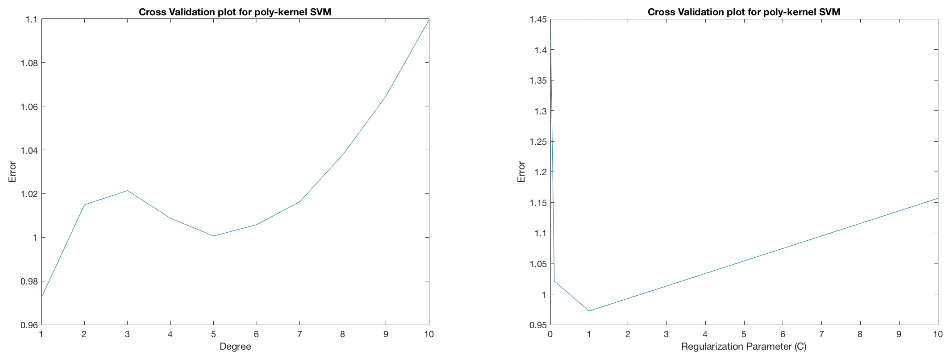

Figure 3: CV Error vs Degree, CV Error vs Regularization Parameter

The next attempt was to train an SVM model with RBF kernel.

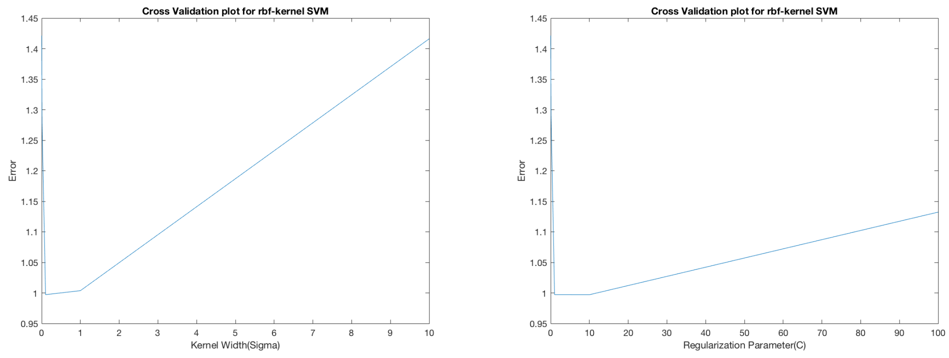

Figure 4: CV Error vs Kernel Width, CV Error vs Regularization Parameter

Again we tuned two parameters - Kernel width (σ) and Regularization parameter (C). A two-dimensional 5-fold cross validation was performed. Kernel Width of $\sigma = 0.1$ and regularization parameter C $= 10$ corresponded to the smallest cross-validation errors.
Cost on validation set: **0.955**
The Linear Kernel performed better than the higher dimensional kernels because we are already in very high dimensional space with not enough data points (the number of data points is similar to the number of features). In such a scenario fitting a high order kernel would be equivalent to fitting a very complicated function to a few data points and hence would result in over-fitting.

- **Principal Component Analysis**

  Cost obtained with PCA with 10 principal components: **1.4493**
  This cost reduces as the number of principle components are increased but the code starts to take very long time. However, we are never able to achieve a cost as good as logistic regression. **The reason for this is explained in section 3.2**.

5 Summary and Conclusion

Logistic Regression with TF-IDF - Cost = 0.9161
Naive Bayes - Cost = 0.9447
K-nearest Neighbors - Cost = 1.2 (cross validation cost)

Note: Our reasoning for these costs is mentioned in individual sections above.

After all the methods we attempted, it appears that the least cost was obtained using the Logistic Regression model with logarithmic TF-IDF. It gave us our minimum cost of 0.9161. We noticed that the pre-processing of the data also did consistently reduce our cost, since it was helping normalization. However, we did try our best to watch out to not over-fit. We feel a more optimal model could have been reached with an ensemble possibly of SVM, Logistic Regression and Naive Bayes.

6 References

[1] Sida Wang and Christopher D Manning. "Baselines and Bigrams: Simple, Good Sentiment and Topic Classification."
[2] Vishal A. Kharde and S.S. Sonawane. 2016. "Sentiment Analysis of Twitter Data: A Survey of Techniques". International Journal of Computer Applications (0975 8887) Volume 139 No.11
[3] Apoorv Agarwal,Boyi Xie,Ilia Vovsha,Owen Rambow and Rebecca Passonneau.2011."Sentiment Analysis of Twitter Data".LSM '11 Proceedings of the Workshop on Languages in Social Media Pages 30-38
[4] Chen, Kewen, et al. "Turning from TF-IDF to TF-IGM for term weighting in text classification." Expert Systems with Applications 66 (2016): 245-260.
[5] Chen, Jingnian, et al. "Feature selection for text classification with Nave Bayes." Expert Systems with Applications 36.3 (2009): 5432-5435.

7 Appendix

We made an attempt to improve the performance of Naive Bayes by generating bi-grams. Here is the code for generating bi-grams:

```
b={};
for i =1:length(train_raw)
    train_raw{i} = erase(train_raw{i},"<user>");
    train_raw{i} = erase(train_raw{i},stopwords);
    %Idx = regexp(train_raw{i}, '[^,@#%$_+="]');
    Idx = regexp(train_raw{i}, '[A-Z a-z]');
    train_raw{i} = train_raw{i}(Idx);
    train_raw{i} = strtrim(train_raw{i});
    train_raw{i} = regexprep(train_raw{i},' +',' ');
    out=regexp(train_raw{i},'\s+','split');
    b{i} = cellfun(@(x,y) [x ' ' y],out(1:end-1)', out(2:end)','un',0)';
end

big=[];
for i =1:length(train_raw)
    out=regexp(train_raw{i},'\s+','split');
    big = horzcat(big,cellfun(@(x,y) [x ' ' y],out(1:end-1)', out(2:end)','un',0)');
end

bigrams = unique(big);

bi_ind = randperm(length(bigrams),50000);
bigrams = bigrams(bi_ind);
save('bigr.mat','bigrams');


% populate bigram matrix
big_train = [];
for i = 1:length(b)

[~,~,ind] = unique(vertcat(bigrams',b{i}'));
bigrams_lb = ind(1:numel(bigrams')); %// label bigrams1
words1_lb = ind(numel(bigrams')+1:end);  %// label words1
counts = sum(sparse(bsxfun(@eq,bigrams_lb,words1_lb')),2);
out = sparse(counts');
big_train = vertcat(big_train,out);
end
```

Code to generate t-SNE plots:

```
f = full(X_train_bag);
rng default % for reproducibility

Yid = find(Y_train == 1| Y_train == 2);
Y=Y_train(Yid);
X = X_train_bag(Yid,:)
f2 = full(X);

T = tsne(f2);

gscatter(T(:,1),T(:,2),Y)
title("Joy-Sadness")
```