• **Project title** : **Classification d'articles ArXiv Kaggle Report**

• **Team name** : Sai Aravind

• **Introduction**:

Given the abstracts of the scientific articles of 15 unique classes, we need to build a classifier to classify the new unseen abstracts of scientific articles in to these 15 classes. For this problem statement a voting classifier of MNB, Linear regression, Linear SVC with optimal hyperparameters was used to achieve an accuracy of 80.6 on the test set given in Kaggle.

• **Feature Design:**

Since this is textual data with 500 samples in each class with a total of 7500 samples. First, the noise in the textual data should be removed to reduce the unnecessary vocabulary being present when the feature set is being constructed. Hence preprocessing text should be done.

Steps in preprocessing involve:

1)Remove the punctuation marks and noise characters(To remove noisy characters)

2) lower the string (To avoid the different representation of uppercase and lowercase of a word in the vocab)

3)remove stop words and digits(To decrease the vocab and also to remove predominant words present in all samples which doesn't add value for classification)

4) Lemmatizing(To build the vocab of words in its reduced form and to avoid different forms of the same word)

Feature design :

Once the preprocessing is complete , now the features must be generated from this cleaned dataset.

For this we use the tf-idf vectorizer from sklearn to build the tf-idf vector for each sample in the training set.

For the purpose of adding more context in to the vector , ngrams =2 was used so that some order of sentence can be retained.

L2 norm was applied in the tf-idf vectorizer to normalize the vector.

Feature selection:

Vocab which is repeated in most of the sentences like stop words are already removed , to further reduce the size of feature set , words which were repeated only once were removed and words in almost 90% of documents were removed as well using the max_df and min_df parameters in tfidf vectorizer api.

As these features are too uncommon to learn a pattern or too common to make out a pattern. Hence these words don't contribute to classification.

• **Algorithms**:

The learning algorithms used were Multinomial NB, LinearSVC, Logistic regression, Voting classifier and Complement NB

Multinomial NB :

Multinomial NB implements the naive Bayes algorithm in the multinomial setting , and is generally used in text classification .In this we calculate the probability of that sentence being in each class and then predict the class with max probability. Here the parameters to tweak are alpha-smoothing parameter

LinearSVC:

The Linear Support Vector Classifier (SVC) method applies a linear kernel function to perform classification. If we compare it with the SVC model, the Linear SVC has additional parameters such as penalty normalization which applies 'L1' or 'L2' and loss function. The kernel method cannot be changed in linear SVC, because it is based on the kernel linear method.

Logistic regression:

A linear classification algorithm which uses the non -linear transform at the end to make the output lie between 0 to 1.This has the option of OVR or multinomial setting for the multi class classification. It has the options of applying regularization and different solver for getting to the best possible weights.

Voting classifier:

The idea behind the Voting classifier is to combine conceptually different machine learning classifiers and use a majority vote or the average predicted probabilities (soft vote) to predict the class labels. Such a classifier can be useful for a set of equally well performing model in order to balance out their individual weaknesses.

Complement NB:

CNB is an adaptation of the standard multinomial naive Bayes (MNB) algorithm that is particularly suited for imbalanced data sets. Specifically, CNB uses statistics from the *complement* of each class to compute the model's weights. The inventors of CNB show empirically that the parameter estimates for CNB are more stable than those for MNB. Further, CNB regularly outperforms MNB (often by a considerable margin) on text classification tasks.

• **Methodology**:

1) Train-test split-
   Since there are only 7500 samples, I split the data in to 80:20 for train and dev-test. Here I don't use a validation set, instead I go ahead with the k -fold stratified cross validation on the train data and train the model on train split to measure the average performance of the model.

2) Vectorization-

Here I am using the tf-idf vectorizer with ngram_range of (1,1) ,(1,2) and (1,3) to select the bect form of vectorization which will increase the no .of features (relevant since ngrams contains word order which might have the context information)

3) Model training-
In this phase , I chose multiple algorithms like MNB , Linear SVC , Logistic regression and then plotted the accuracy bar plots of each of these algos in the default setting with the 5-fold stratified cross validation and observed which algorithm is doing better in terms of average accuracy.

4) Model Analysis-
Now, take the best performing algorithms in my case the MNB and logistic regression with average accuracy of 80.By using these models I try to see the performance of models on the dev-test data. Here I predict on the test data and then view the accuracy report and confusion matrix to see which classes are doing better and which are mostly being misclassified. This gives an idea on which part of model should be improved to see that model will perform better on test set as well

5) Model Tuning-
Once I had the pipeline of vectorization and model training. Now I try to find the best parameters of the vectorizer and the model. First wrt to vectorizer, I found that ngram range of (1,1) performed better rather than the ngram range –(1,2) or (1,3).So I fixed the quantity to ngram range (1,1).
Now with that fixed, next was to find the best parameters for the models - logistic regression,Multinomial Naïve Bayes and linear SVC. For this I used the grid search algorithm for choosing the best parameters for each algo one at a time. For MNB it was alpha, so it was straight forward.
For Logistic regression it had C, Solver, penalty and max_iter parameters. The grid search was done for this and the best parameters were found=d to be c=4.8, solver=newton-cg, penalty=l2.
For Linear SVC I have tweaked the parameters C, penalty , max_iter, loss to find the best possible setting using the grid search.
For the grid search a 5 fold cross validation approach has been used.

6) Measuring tuned model performance-
Now using this fine-tuned model, predictions on the dev-test set are made and this is compared with the default setting models. This shows that it this finetuning helped with respect to the misclassified classes not by a huge margin but an improvement from the default settings.

7) Kaggle Test predictions-
Now that I have confidence on my model, now I want to see the leaderboard score and how it perform on the unseen dataset of 15,000 samples. Here I made the predictions using my best models of MNB ,Linear SVC and LG individually to see how I would fare on the leader board. The mNB achieved a better accuracy compared to LG and Linear SVC.

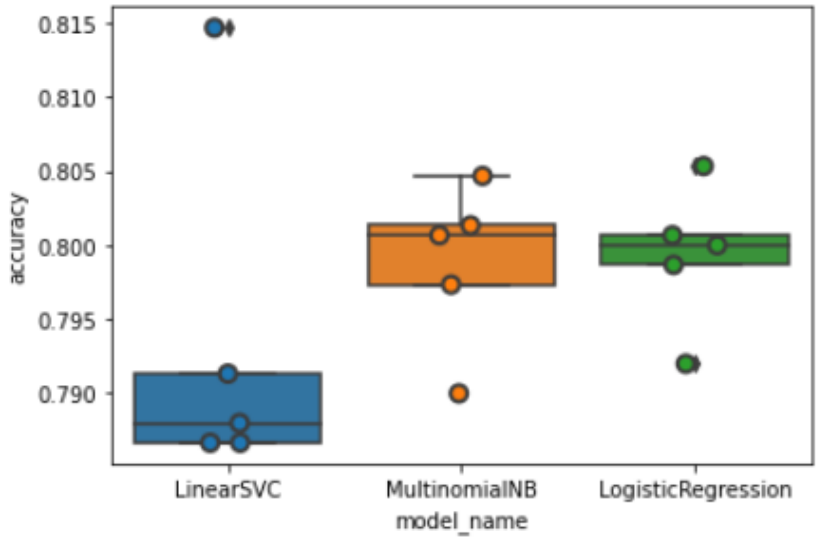
8) Ensembling-
   I have read that instead of using models individually, combining the output usually gives better accuracy. To test this out, I used the simplest ensembling algorithm i.e., the voting Classifier to combine my best algorithms MNB(0.2) ,CalibratedClassifierCV(linear SVC)* and Logistic regression(best params) using the soft voting methodology. This has improved the avg accuracy in the cross validation of train data. The results were then observed on the dev-test which showed a small increase in accuracy. I combine the train and dev-test data to train the voting classifier to predict on the Kaggle unseen test set of 15000 samples

9) Testing Out different ideas (hierarchical classification)-
   I observed that mostly the misclassification is among the astro-ph classes (5 of them). So I combined all astro-ph related classes in to one class and then build a 2 level classification mechanism. One classification for classes (with combined astro-ph classes) and then whichever is classified as astro-ph it is given to another classification model which will classify which of the astro-ph sub classes it falls under. I tried this out in the same way .i.e., find best params at each level of classification, put it in the voting classifier at each level.
   Here since I combined the classes, there is an imbalance between classes. To counter that I used Complement NB instead of MNB and also used 'balanced' as input to class_weight param in algorithms
   Now I have two classifiers voting1 for classes (all classes other astro-ph and combined astro-ph classes)
   Voting2 – for only astro-ph sub classes.
   This classifier gave me a small increment in classification accuracy wrt the misclassified classes .this was observed on the dev-test set. This idea didn't result in a increase in Kaggle test accuracy score. This suggests that this is trying to overfit or was not able to generalize well.

10) Final model-
    After trying out different parameters and ideas as described in the steps above , the best model to achieve the test score of 80.6 on Kaggle test set was the simple voting classifier with best params of Logistic , MNB and linear SVC (with tf-idf vectorization of ngram(1,1) and l2 norm). All other models even though showed good accuracy on dev-test , it was mostly because it was overfitting on 1 or 2 classes and so it was failing badly in other classes. This final model was able to strike a balance between the classes by not titling towards specific classes.

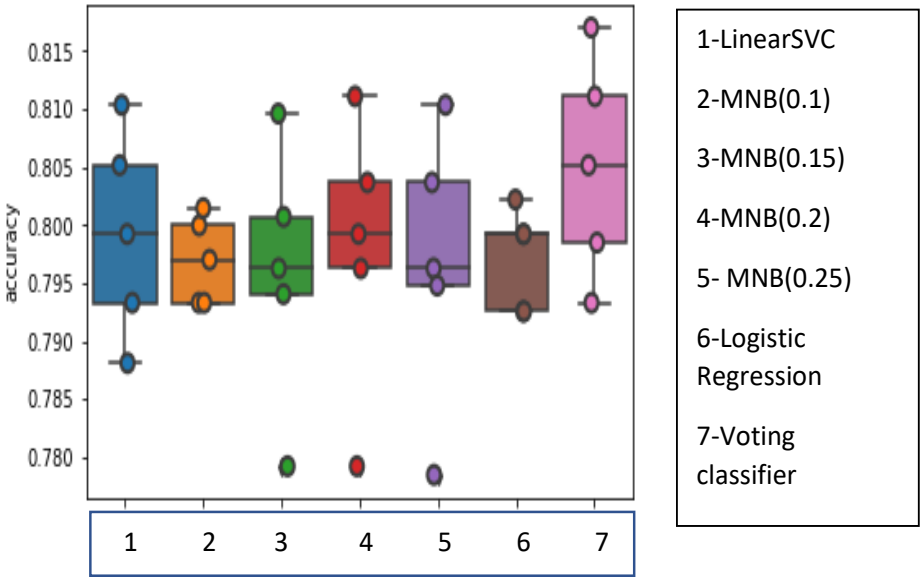
• **Results**:

Default scenario setting: default parameters in algorithms, Tf-idf: ngram range -(1,1) [5-fold] :



In this default scenario setting, as it can be seen in the above figure, we have average accuracies of models ranging between 0.78 to 0.8. The best performing models were the MNB and Logistic regression with an average of 0.8

Best parameter setting (after grid search on individual algorithms) tf-idf ngram(1,1) [5-fold] :



From the above bar plots, it is easy to visualize that voting classifier was performing the best and that the grid search parameters we got for individual worked better compared to the default setting scenario. The 7 represents the final model used for predicting on the Kaggle test set.This was achieved for a cross validation of 5 fold.

Detailed analysis of results:

From the confusion matrix and classification report in appendix, the main culprits for decreasing the accuracy were mainly the astro-ph classes.The worst performing models were not able to do a good job here. whereas the hierarchical classification approach was able to perform well in the level2 , but it was simply overfitting on it due to less samples .Hence it couldn't perform well on the actual test set.

The final model of Voting classifier was able to get a better precision and recall in case of astro-ph classes compared to all other models that were tried. So , its performance was better on the unseen test set.

• Discussion:

Pros:

1) simplistic ML algorithms which are easy to infer in terms of prediction (see the best words for a particular class)

2) Ensembled method - voting classifier was used to get the best of all 3 algorithms used for classification

3) relatively less compute intensive which helped in faster experimentation

Cons:

1)The hierarchical approach was overfitting, so it couldn't perform better on actual test set

2)No boosting algorithm has been used which was another way of ensembling

3)The feature selection was not good enough even though penalty was applied, rigorous feature selection would have made an impact.

4)Lacks context , since it only looks at words and has no order.

Ideas for improvement:

1)Analyze the top words for classification per class and then try to do feature section based on it with ngrams

2)Boosting approach hasn't been tried , so boosting algorithms might work well in case of the misclassified classes

3)Testing out word vectors (like fast text or USE) and then building LSTM / classification layer

4) Use the pretrained text models to get embeddings and give it to the classification layer to get the predictions. Finetuning the transformer for text classification might work well, because the traditional ML methods weren't able to get the context and just used the presence of words.

5)Use the readily available text classifier in libraries like flair and spacy for advanced and quick implementation of embedding based deep learning methods.

• **References**

https://scikit-learn.org/stable/modules/ensemble.html#voting-classifier

https://scikit-learn.org/stable/modules/naive_bayes.html#multinomial-naive-bayes

https://towardsdatascience.com/logistic-regression-model-tuning-with-scikit-learn-part-1-425142e01af5

https://www.kaggle.com/abhishek/approaching-almost-any-nlp-problem-on-kaggle

https://towardsdatascience.com/hyperparameter-tuning-c5619e7e6624

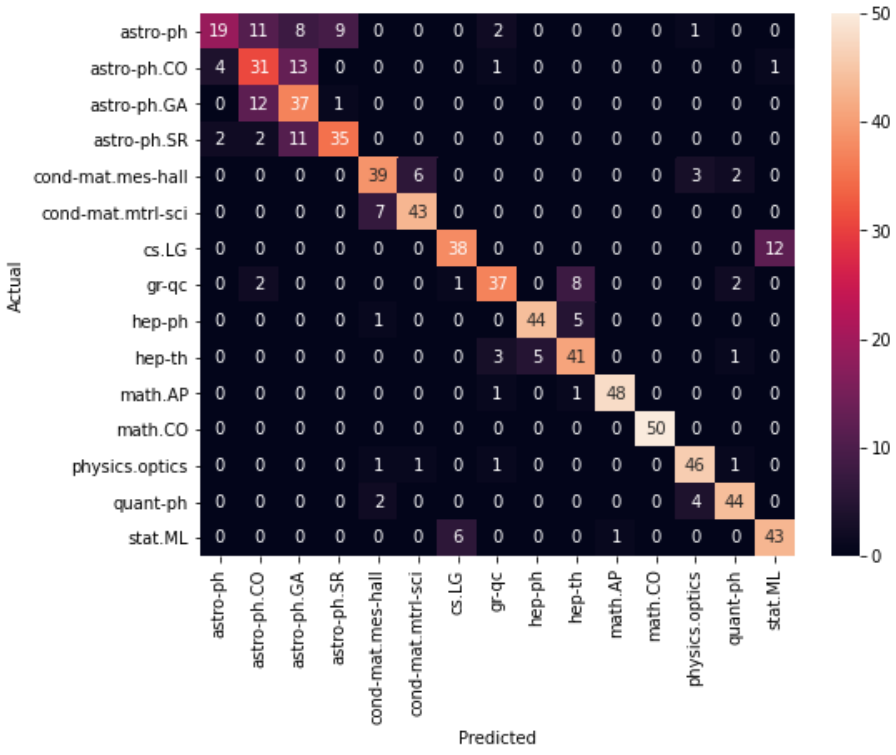
https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/

https://www.datatechnotes.com/2020/07/classification-example-with-linearsvm-in-python.html

• **Appendix**

*- used calibrated Linear SVC for the purpose of getting predict_proba , since I was using the voting method as soft in voting classifier

Confusion matrix for only 10% of dev-test data



Classification Report:

	precision	recall	f1-score	support
0	0.76	0.38	0.51	50
1	0.53	0.62	0.57	50
2	0.54	0.74	0.62	50
3	0.78	0.70	0.74	50
4	0.78	0.78	0.78	50
5	0.86	0.86	0.86	50
6	0.84	0.76	0.80	50
7	0.82	0.74	0.78	50
8	0.90	0.88	0.89	50
9	0.75	0.82	0.78	50
10	0.98	0.96	0.97	50
11	1.00	1.00	1.00	50
12	0.85	0.92	0.88	50
13	0.88	0.88	0.88	50
14	0.77	0.86	0.81	50
accuracy			0.79	750
macro avg	0.80	0.79	0.79	750
weighted avg	0.80	0.79	0.79	750

:

{0: 'astro-ph',

1: 'astro-ph.CO',

2: 'astro-ph.GA',

3: 'astro-ph.SR',

4: 'cond-mat.mes-hall',

5: 'cond-mat.mtrl-sci',

6: 'cs.LG',

7: 'gr-qc',

8: 'hep-ph',

9: 'hep-th',

10: 'math.AP',

11: 'math.CO',

12: 'physics.optics',

13: 'quant-ph',

14: 'stat.ML'