# IFT 6390 Kaggle Competition I - Report

Arka Mukherjee

TOTAL POINTS

**87 / 90**

QUESTION 1

## 1 Competition 67 / 70

+ **55 pts** Correct

+ **67** Point adjustment

QUESTION 2

## Report 20 pts

### 2.1 General formatting 2 / 2

✓ + **1 pts** statement of original work

✓ + **1 pts** main content is within 6 pages

+ **0 pts** none of the above

### 2.2 Feature extraction description 4 / 4

✓ + **2 pts** bag of words features (incl. additional "tricks" e.g. using stop words either from NLTK or handcrafted)

✓ + **2 pts** tf-idf features

+ **2 pts** word embedding

+ **2 pts** n-grams (n>1)

+ **2 pts** sentence embedding (ala BERT)

+ **2 pts** other features

✓ + **1 pts** more advanced bow (e.g. stemming)

+ **0 pts** no description

### 2.3 Algorithms 6 / 6

✓ + **3 pts** naive bayes/complement NB

✓ + **3 pts** logistic regression

✓ + **3 pts** svm

+ **3 pts** other linear methods

✓ + **3 pts** neural networks

+ **3 pts** decision trees/random forests

✓ + **3 pts** other ensemble methods

+ **3 pts** KNN

+ **3 pts** other model

+ **0 pts** no description

### 2.4 Methodology 4 / 4

✓ + **4 pts** cross validation for hyperparameter tuning

+ **0 pts** no mention/use of cross validation

+ **1 pts** vague mention of cross validation, but no description on how hyperparameter tuning was done

### 2.5 Discussion of results 4 / 4

✓ + **4 pts** clear discussion of results

+ **1 pts** superficial discussion of results

+ **0 pts** no discussion of results



gradescope

1 Competition **67 / 70**

+ **55 pts** Correct

+ **67** **Point adjustment**



gradescope

**Project Title: Predicting the categories for unseen abstracts of Arxiv papers, Kaggle Competition**

**"Data Competition – I", IFT - 6390**

**Team Name: Single Member**

**Member: Arka Mukherjee**

**Professional MSc Student, Mila, Fall 2020**

**UdeM Matricule: 20182625**

## Problem Introduction:

In this Kaggle Competition, I have attempted to build an effective Text Classification Algorithm that works well on unseen data. The given dataset contains of two columns, one 'Abstract' and one 'Category'. The abstracts are basically abstracts of several research papers. The categories are 15 unique categories (eg: astro-ph, hep-ph, math.CO, cs.LG, etc).

To beat the required baselines, I have divided my work into the following main stages:

1. Data cleaning and pre-processing.
2. Model imports and Bag of Words/vectorized model creation (BoW/Count Vectorizer/TFIDF Vectorizer/Data Pipelines with a TFIDF Transformer).
3. Hyperparameter Searches and Automated Evaluation.
4. Model Training and Cross-Validation.
5. Predicting over the test.csv dataset.
6. Saving the result, and testing on Kaggle.

The goal here is to not just fit the training data well, but to work decently even with real life abstracts. Because of that, I tried to focus more on Cross Validation results/outcomes before testing the outcome out on Kaggle. On top of that, there were only two submission attempts per day, which made me want to test my model more before uploading a submission on the portal. To process the raw data, I made use of regular expressions (regex), and the NLTK library. I tried numerous algorithms out for this project, and I had started out with LSTM (after reading a lot of papers about it being effective), and realized that LSTMs with even a lower capacity and Sigmoid Activation Functions in hidden layers can overfit very badly, and reach a training accuracy of 98%+ and a cross validation accuracy of 66%, which was well below the Naïve Bayes' baselines. After this, I decided to go for relatively simpler models. I started out with Gaussian Naïve Bayes' using SKlearn, and I got a moderately reasonable accuracy. Then, I tried several other models – Multinomial Naïve Bayes, Bernoulli Naïve Bayes, Logistic Regression, several types of trees including Random Forest, Decision Tree classifiers, Extra Tree Classifiers. I had also explored Stochastic Gradient Classifier, a Multi-Layer Perceptron, an Adaboost Classifier, Gradient Boosting Classifier and an SVM Classifier with different kernels (Linear Kernel, Polynomial Kernel, and an RBF Kernel).

Furthermore, I had implemented a Hyperparameter search for few of the models in an attempt to get better tuning parameters.

The final model which worked the best for me a combination of Multinomial NB, Gaussian NB, SVM with Linear Kernel, Bernoulli Naïve Bayes, Logistic Regression, MLP, SGD Classifier and Gradient Boosting Classifier. The learning rate for MLP and the smoothing parameter for the Bayes' algorithms were tuned using results obtained from an exhaustive grid search technique. The final ensembling algorithm used was a hard-voting algorithm (the default in SKLearn), and custom weights (set based on manual hyperparameter search in our project). The final validation accuracy of my algorithm was 80.3%, and my training dataset accuracy was approximately 98.96%. Meanwhile, the Naïve Bayes from Scratch algorithm had obtained a 78.933% accuracy on unseen data (an unbiased statistical estimate).

## 2.1 General formatting 2 / 2

✓ + **1 pts** statement of original work

✓ + **1 pts** main content is within 6 pages

+ **0 pts** none of the above



gradescope

## Feature Design:

In this section, we go over how the dataset was optimised for a good fit for machine learning algorithms. It is often said that feature engineering is the most important part of data science, so I applied and tested a variety of feature engineering methods for my project.

Let us go step by step, and explore what all I have tried out in this project.

1. **Converting every string to lowercase:** Our given dataset is a dataset of strings, and converting everything to lower-case makes it more unform. There should be no distinction between the words "Facebook" and "facebook", as contextually, they would mean the same. For example, consider the two sentences:

   *Sentence 1:* Facebook is a good social media website.
   *Sentence 2:* The social media website called facebook is good.

   Even though both the sentences are (almost) grammatically correct, except the proper noun capitalization, the contexts are the same. I would not want my Machine Learning algorithm to treat these cases differently.
2. **Replace multiple spaces with a single space:** Often, human errors result in a double press of the space bar, which can cause duplicate features. I replace them using basic regular expressions.
3. **Removing emojis or emoticons:** Here, I use a set of predefined Emoji Hex values, and remove them from my Pandas DataFrame. They are not useful in text classification scenarios like these.
4. **Unidecode our text:** In some abstracts, there might be data that is in a different format, and Python may misunderstand the data as "???". I have converted all of them to string using the unidecode library.
5. **Removing StopWords from arxiv Abstracts:** Stopwords are not usually playing a key role in Machine Learning tasks such as these, and they are commonly existing words in the language. I have used the NLTK package to download them, and then remove from the abstract strings in the column.
6. **Lemmatization and Stemming:** These techniques are usually used to shorten words to a format where the construct of language and grammar does not apply anymore. We shorten words to a root word. This has been again done using NLTK.
7. **Spelling Correction:** I have imported a spell-checker library (from spellchecker import SpellChecker) to fix basic typos of all words of abstract.
8. **Remove HTML from Strings:** I have removed all HTML from the abstract, as it was obfuscating certain categories for the machine learning models I have tested.
9. **Removal of rarest and most common words:** Words which are not too frequent and very frequent do not contribute a lot. Hence, I have removed them (top 50 ones).

The above steps reduce the numbers of features that our vectorized model will make use of, and it is an important part of Feature Engineering in Machine Learning. **For the Naïve Bayes from scratch component, I have used all of them except the ones using NLTK and external libraries.**

*Vectorization:*

I had started out with a Count Vectorizer in SKLearn to beat the first and second baselines. I chose this approach because it internally uses Scipy's Sparse Matrices module, which would make this approach more computationally efficient. We also do not need to give the API a priori document for ML analysis, which makes it very easy to use. We can also pass the binary flag or the stop words parameter for robust data cleaning. For other baselines, the TFIDF (Term Frequency Inverse Document Frequency) with IDF smoothening seemed to work better. The CountVectorizer resorts to simply counting word frequencies, and the TFIDF algorithm, the value increases proportionally to count, but is inversely proportional to frequency of the word in the corpus; that is the inverse document frequency (IDF) part[1]. The TFIDF Vectorizer was better at beating other more complex baselines. However, both TFIDF and CountVectorizer had high dimensionalities. For faster model fitting/training, I had deployed my model on AWS EC2 Instances.

Furthermore, I had also noticed a significant performance improvement if I used a Data Pipeline. I had created a Data Pipeline using a TFIDF Vectorizer, and a TFIDF Transformer on the DataFrame's abstract. A pipeline object enables us to integrate several well-crafted data vectorized mechanisms (normalization, feature analysis, vectorization) into one. Pipelines are basically DAG (Directed Acyclic Graphs), which can club many techniques into one unit. I had also run a Grid Search CV over a few ranges for n-gram values (could not try many hyperparameters due to high distributed memory consumption for AdaBoostClassifiers and the MultiLayerPerceptron), and my model had given me (1,2) for the n-gram range.

Our final feature vector had the shape of (7500, 29266). If I did not do a complete feature vectorization, I would have observed that the number of columns in the shape of my feature vector would be 75000+. This further illustrates that Feature Engineering is one of the most important phases of Data Engineering or Machine Learning Engineering. For the Naïve Bayes from Scratch algorithm, I have designed a frequency based BoW (Bag of Words) data model.

## Main Algorithms used in the project:

1. **Multinomial Naïve Bayes:** This algorithm internally finds log probabilities in data using the Bayes Theorem (with the Naïve Assumption of mutual independence) and assuming the data has a multinomial distribution.
2. **Gaussian Naïve Bayes:** Here, we calculate the probabilities using the Bayes' Theorem with the Naïve Assumption, and a Gaussian Kernel is used to compute the probability in the prediction case and it supports continuous data.
3. **Bernoulli Naïve Bayes:** This is based on Bernoulli's Distribution, we can apply the Bayes' probability formula directly and enable Laplace Smoothing. It accepts only binary values (0/1) which correspond to a failure or a success.
4. **SVM Classifier with all kind of kernels:** These models work by the principal of Maximum Margin and weight optimization. The loss function is a function of Margin, and we calculate the maximum margin for the classifier, by minimizing the loss.
5. **Logistic Regression:** This algorithm uses a sigmoid or logistic function to find maximum probability of a class (p for each class is between 0 and 1).
6. **Adaboost Classifier:** An AdaBoost classifier[2] is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.
7. **Decision Trees:** These models split our dataset according to a set of condition using splitting points. We usually regularize a Decision Tree by Pruning branches or paths.
8. **Extra Trees:** Similar to Decision Trees and Random Forest (which is basically an ensembled Decision Tree), but the tree building and splitting point selection is completely randomized.
9. **Stochastic Gradient Descent (SGD) Classifier:** These are linear classifiers which use batch or stochastic gradient descent for training the weights.
10. **Multi Layer Perceptron (MLP):** This is a fundamental ANN with hidden layers, and the weights/units are trained using an algorithm like backpropagation, which is reminiscent of how our brain learns.
11. **Voting Classifier:** This is an ensemble algorithm that does a majority vote among classifiers. There are two kinds of Voting Classifiers – Soft Voting Classifiers, and Hard Voting Classifiers.
12. **LSTM RNN[6]:** It is a kind of ANN that has both feedforward and feedback connections. This can been implemented using Keras. I have tried the Dropout technique to improve overfitting.

## Methodologies used:

**Cross-Validation**: In our project, we have divided our dataset into two components – one training set with 70% of the data, and one cross validation set, with 30% of the data. I tested it only once and I had only one split, which makes this an example of Holdout Cross Validation. Randomized splitting has been done here[3].

## 2.2 Feature extraction description 4 / 4

✓ + **2 pts** **bag of words features (incl. additional "tricks" e.g. using stop words either from NLTK or handcrafted)**

✓ + **2 pts** **tf-idf features**

+ **2 pts** word embedding

+ **2 pts** n-grams (n>1)

+ **2 pts** sentence embedding (ala BERT)

+ **2 pts** other features

✓ + **1 pts** **more advanced bow (e.g. stemming)**

+ **0 pts** no description



gradescope

Furthermore, I had also noticed a significant performance improvement if I used a Data Pipeline. I had created a Data Pipeline using a TFIDF Vectorizer, and a TFIDF Transformer on the DataFrame's abstract. A pipeline object enables us to integrate several well-crafted data vectorized mechanisms (normalization, feature analysis, vectorization) into one. Pipelines are basically DAG (Directed Acyclic Graphs), which can club many techniques into one unit. I had also run a Grid Search CV over a few ranges for n-gram values (could not try many hyperparameters due to high distributed memory consumption for AdaBoostClassifiers and the MultiLayerPerceptron), and my model had given me (1,2) for the n-gram range.

Our final feature vector had the shape of (7500, 29266). If I did not do a complete feature vectorization, I would have observed that the number of columns in the shape of my feature vector would be 75000+. This further illustrates that Feature Engineering is one of the most important phases of Data Engineering or Machine Learning Engineering. For the Naïve Bayes from Scratch algorithm, I have designed a frequency based BoW (Bag of Words) data model.

## Main Algorithms used in the project:

1. **Multinomial Naïve Bayes:** This algorithm internally finds log probabilities in data using the Bayes Theorem (with the Naïve Assumption of mutual independence) and assuming the data has a multinomial distribution.
2. **Gaussian Naïve Bayes:** Here, we calculate the probabilities using the Bayes' Theorem with the Naïve Assumption, and a Gaussian Kernel is used to compute the probability in the prediction case and it supports continuous data.
3. **Bernoulli Naïve Bayes:** This is based on Bernoulli's Distribution, we can apply the Bayes' probability formula directly and enable Laplace Smoothing. It accepts only binary values (0/1) which correspond to a failure or a success.
4. **SVM Classifier with all kind of kernels:** These models work by the principal of Maximum Margin and weight optimization. The loss function is a function of Margin, and we calculate the maximum margin for the classifier, by minimizing the loss.
5. **Logistic Regression:** This algorithm uses a sigmoid or logistic function to find maximum probability of a class (p for each class is between 0 and 1).
6. **Adaboost Classifier:** An AdaBoost classifier[2] is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.
7. **Decision Trees:** These models split our dataset according to a set of condition using splitting points. We usually regularize a Decision Tree by Pruning branches or paths.
8. **Extra Trees:** Similar to Decision Trees and Random Forest (which is basically an ensembled Decision Tree), but the tree building and splitting point selection is completely randomized.
9. **Stochastic Gradient Descent (SGD) Classifier:** These are linear classifiers which use batch or stochastic gradient descent for training the weights.
10. **Multi Layer Perceptron (MLP):** This is a fundamental ANN with hidden layers, and the weights/units are trained using an algorithm like backpropagation, which is reminiscent of how our brain learns.
11. **Voting Classifier:** This is an ensemble algorithm that does a majority vote among classifiers. There are two kinds of Voting Classifiers – Soft Voting Classifiers, and Hard Voting Classifiers.
12. **LSTM RNN[6]:** It is a kind of ANN that has both feedforward and feedback connections. This can been implemented using Keras. I have tried the Dropout technique to improve overfitting.

## Methodologies used:

**Cross-Validation**: In our project, we have divided our dataset into two components – one training set with 70% of the data, and one cross validation set, with 30% of the data. I tested it only once and I had only one split, which makes this an example of Holdout Cross Validation. Randomized splitting has been done here[3].

| Algorithms on the TFIDF Vectorizer + TFIDF Transformer | Cross Validation Split | Accuracy (%) |
|---|---|---|
| Multinomial Naïve Bayes (Alpha = 0.15) | Training Accuracy | 95.7 |
| | Testing Accuracy | 78.973 |
| Gaussian Naïve Bayes | Training Accuracy | 99.66 |
| | Testing Accuracy | 62.8 |
| SVM with Linear Kernel | Training Accuracy | 99.516 |
| | Testing Accuracy | 77.46 |
| Adaboost Classifier (alpha = 0.1, n_estimator = 400) | Training Accuracy | 53.48 |
| | Testing Accuracy | 48.53 |
| SGD Classifier | Training Accuracy | 99.150 |
| | Testing Accuracy | 77.133 |
| Logistic Regression | Training Accuracy | 93.533 |
| | Testing Accuracy | 77.866 |
| MLP Classifier | Training Accuracy | 1.0 |
| | Testing Accuracy | 77.1 |
| Decision Tree Classifier | Training Accuracy | 1.0 |
| | Testing Accuracy | 53.1 |
| Extra Tree Classifier | Training Accuracy | 1.0 |
| | Testing Accuracy | 75.4 |
| LSTM RNN | Training Accuracy | 98.38 |
| | Testing Accuracy | 66.2 |
| **Naïve Bayes from Scratch (Kaggle Q4)** | Training Accuracy | 96.2 |
| | Testing Accuracy | 78.93 |
| Bernoulli Naïve Bayes | Training Accuracy | 92.48 |
| | Testing Accuracy | 76.2 |
| **Final Ensemble Algorithm** | Training Accuracy | 98.96 |
| | Testing Accuracy | 80.3 |

**Distribution Choice for Naïve Bayes**: I personally used three models of NB in my final ensemble algorithm, and to find optimal values for Alpha, I used the Grid Search Algorithm. My final Alpha value for the MultiNB model was around 0.15 (and Bernoulli at 0.10), which seemed optimal based on my hyperparameter search. We know that Multinomial NB makes the assumption that the data is multinomially distributed.

**Weighted Hard Voting in the Ensemble Algorithm**: I realized that when I fit the data individually for all the models, some models were more effective than others. Some overfit (eg: MLP and Adaboost), and some underfit (e.g.: Gaussian NB). I tested a number of voting combination in hard voting and soft voting, and hard voting seemed to work better for me, which is why I sticked to it for my submission[4].

**Regularization:** My Logistic Regression Model had the penalty set to 'L2' (SKLearn default)[5]. I noticed that the model with L2 performed better than an L1 or ElasticNet penalty. My MLP Model's penalty value alpha was increased a little bit too to improve overfitting (to 0.12).

**Activation Function:** In my initial LSTM tests, I had tried both the Softmax Activation Function and the Sigmoid Activation Function, and Sigmoid seemed to perform better.

## Results:

I had tested the following classical Machine Learning Algorithms for this project (given in the table to the top left).

From these classical Machine Learning Algorithms, I have selected the ones that not only perform the best, but have a reasonable bias-variance tradeoff. We have also tuned the hyper-parameter alpha (which is the additive smoothing parameter) for Multinomial Naïve Bayes, for the Bernoulli Naïve Bayes Algorithm and for the Gaussian Naïve Bayes Algorithm. Furthermore, I will also show how results vary when I change the Activation Functions for my LSTM RNN, and switch between different optimizers in Keras. First, let us show how the validation accuracy for Multinomial Naïve Bayes and Bernoulli Naïve Bayes varies with alpha (Grid Search):

## 2.3 Algorithms 6 / 6

- ✓ + **3 pts** naive bayes/complement NB
- ✓ + **3 pts** logistic regression
- ✓ + **3 pts** svm
- + **3 pts** other linear methods
- ✓ + **3 pts** neural networks
- + **3 pts** decision trees/random forests
- ✓ + **3 pts** other ensemble methods
- + **3 pts** KNN
- + **3 pts** other model
- + **0 pts** no description



gradescope

Furthermore, I had also noticed a significant performance improvement if I used a Data Pipeline. I had created a Data Pipeline using a TFIDF Vectorizer, and a TFIDF Transformer on the DataFrame's abstract. A pipeline object enables us to integrate several well-crafted data vectorized mechanisms (normalization, feature analysis, vectorization) into one. Pipelines are basically DAG (Directed Acyclic Graphs), which can club many techniques into one unit. I had also run a Grid Search CV over a few ranges for n-gram values (could not try many hyperparameters due to high distributed memory consumption for AdaBoostClassifiers and the MultiLayerPerceptron), and my model had given me (1,2) for the n-gram range.

Our final feature vector had the shape of (7500, 29266). If I did not do a complete feature vectorization, I would have observed that the number of columns in the shape of my feature vector would be 75000+. This further illustrates that Feature Engineering is one of the most important phases of Data Engineering or Machine Learning Engineering. For the Naïve Bayes from Scratch algorithm, I have designed a frequency based BoW (Bag of Words) data model.

## Main Algorithms used in the project:

1. **Multinomial Naïve Bayes:** This algorithm internally finds log probabilities in data using the Bayes Theorem (with the Naïve Assumption of mutual independence) and assuming the data has a multinomial distribution.
2. **Gaussian Naïve Bayes:** Here, we calculate the probabilities using the Bayes' Theorem with the Naïve Assumption, and a Gaussian Kernel is used to compute the probability in the prediction case and it supports continuous data.
3. **Bernoulli Naïve Bayes:** This is based on Bernoulli's Distribution, we can apply the Bayes' probability formula directly and enable Laplace Smoothing. It accepts only binary values (0/1) which correspond to a failure or a success.
4. **SVM Classifier with all kind of kernels:** These models work by the principal of Maximum Margin and weight optimization. The loss function is a function of Margin, and we calculate the maximum margin for the classifier, by minimizing the loss.
5. **Logistic Regression:** This algorithm uses a sigmoid or logistic function to find maximum probability of a class (p for each class is between 0 and 1).
6. **Adaboost Classifier:** An AdaBoost classifier[2] is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.
7. **Decision Trees:** These models split our dataset according to a set of condition using splitting points. We usually regularize a Decision Tree by Pruning branches or paths.
8. **Extra Trees:** Similar to Decision Trees and Random Forest (which is basically an ensembled Decision Tree), but the tree building and splitting point selection is completely randomized.
9. **Stochastic Gradient Descent (SGD) Classifier:** These are linear classifiers which use batch or stochastic gradient descent for training the weights.
10. **Multi Layer Perceptron (MLP):** This is a fundamental ANN with hidden layers, and the weights/units are trained using an algorithm like backpropagation, which is reminiscent of how our brain learns.
11. **Voting Classifier:** This is an ensemble algorithm that does a majority vote among classifiers. There are two kinds of Voting Classifiers – Soft Voting Classifiers, and Hard Voting Classifiers.
12. **LSTM RNN[6]:** It is a kind of ANN that has both feedforward and feedback connections. This can been implemented using Keras. I have tried the Dropout technique to improve overfitting.

## Methodologies used:

**Cross-Validation**: In our project, we have divided our dataset into two components – one training set with 70% of the data, and one cross validation set, with 30% of the data. I tested it only once and I had only one split, which makes this an example of Holdout Cross Validation. Randomized splitting has been done here[3].

| Algorithms on the TFIDF Vectorizer + TFIDF Transformer | Cross Validation Split | Accuracy (%) |
|---|---|---|
| Multinomial Naïve Bayes (Alpha = 0.15) | Training Accuracy | 95.7 |
| | Testing Accuracy | 78.973 |
| Gaussian Naïve Bayes | Training Accuracy | 99.66 |
| | Testing Accuracy | 62.8 |
| SVM with Linear Kernel | Training Accuracy | 99.516 |
| | Testing Accuracy | 77.46 |
| Adaboost Classifier (alpha = 0.1, n_estimator = 400) | Training Accuracy | 53.48 |
| | Testing Accuracy | 48.53 |
| SGD Classifier | Training Accuracy | 99.150 |
| | Testing Accuracy | 77.133 |
| Logistic Regression | Training Accuracy | 93.533 |
| | Testing Accuracy | 77.866 |
| MLP Classifier | Training Accuracy | 1.0 |
| | Testing Accuracy | 77.1 |
| Decision Tree Classifier | Training Accuracy | 1.0 |
| | Testing Accuracy | 53.1 |
| Extra Tree Classifier | Training Accuracy | 1.0 |
| | Testing Accuracy | 75.4 |
| LSTM RNN | Training Accuracy | 98.38 |
| | Testing Accuracy | 66.2 |
| **Naïve Bayes from Scratch (Kaggle Q4)** | Training Accuracy | 96.2 |
| | Testing Accuracy | 78.93 |
| Bernoulli Naïve Bayes | Training Accuracy | 92.48 |
| | Testing Accuracy | 76.2 |
| **Final Ensemble Algorithm** | Training Accuracy | 98.96 |
| | Testing Accuracy | 80.3 |

**Distribution Choice for Naïve Bayes**: I personally used three models of NB in my final ensemble algorithm, and to find optimal values for Alpha, I used the Grid Search Algorithm. My final Alpha value for the MultiNB model was around 0.15 (and Bernoulli at 0.10), which seemed optimal based on my hyperparameter search. We know that Multinomial NB makes the assumption that the data is multinomially distributed.

**Weighted Hard Voting in the Ensemble Algorithm**: I realized that when I fit the data individually for all the models, some models were more effective than others. Some overfit (eg: MLP and Adaboost), and some underfit (e.g.: Gaussian NB). I tested a number of voting combination in hard voting and soft voting, and hard voting seemed to work better for me, which is why I sticked to it for my submission[4].

**Regularization:** My Logistic Regression Model had the penalty set to 'L2' (SKLearn default)[5]. I noticed that the model with L2 performed better than an L1 or ElasticNet penalty. My MLP Model's penalty value alpha was increased a little bit too to improve overfitting (to 0.12).

**Activation Function:** In my initial LSTM tests, I had tried both the Softmax Activation Function and the Sigmoid Activation Function, and Sigmoid seemed to perform better.

## Results:

I had tested the following classical Machine Learning Algorithms for this project (given in the table to the top left).

From these classical Machine Learning Algorithms, I have selected the ones that not only perform the best, but have a reasonable bias-variance tradeoff. We have also tuned the hyper-parameter alpha (which is the additive smoothing parameter) for Multinomial Naïve Bayes, for the Bernoulli Naïve Bayes Algorithm and for the Gaussian Naïve Bayes Algorithm. Furthermore, I will also show how results vary when I change the Activation Functions for my LSTM RNN, and switch between different optimizers in Keras. First, let us show how the validation accuracy for Multinomial Naïve Bayes and Bernoulli Naïve Bayes varies with alpha (Grid Search):

## 2.4 Methodology 4 / 4

✓ + **4 pts** **cross validation for hyperparameter tuning**

+ **0 pts** no mention/use of cross validation

+ **1 pts** vague mention of cross validation, but no description on how hyperparameter tuning was done



gradescope

| Algorithms on the TFIDF Vectorizer + TFIDF Transformer | Cross Validation Split | Accuracy (%) |
|---|---|---|
| Multinomial Naïve Bayes (Alpha = 0.15) | Training Accuracy | 95.7 |
| | Testing Accuracy | 78.973 |
| Gaussian Naïve Bayes | Training Accuracy | 99.66 |
| | Testing Accuracy | 62.8 |
| SVM with Linear Kernel | Training Accuracy | 99.516 |
| | Testing Accuracy | 77.46 |
| Adaboost Classifier (alpha = 0.1, n_estimator = 400) | Training Accuracy | 53.48 |
| | Testing Accuracy | 48.53 |
| SGD Classifier | Training Accuracy | 99.150 |
| | Testing Accuracy | 77.133 |
| Logistic Regression | Training Accuracy | 93.533 |
| | Testing Accuracy | 77.866 |
| MLP Classifier | Training Accuracy | 1.0 |
| | Testing Accuracy | 77.1 |
| Decision Tree Classifier | Training Accuracy | 1.0 |
| | Testing Accuracy | 53.1 |
| Extra Tree Classifier | Training Accuracy | 1.0 |
| | Testing Accuracy | 75.4 |
| LSTM RNN | Training Accuracy | 98.38 |
| | Testing Accuracy | 66.2 |
| **Naïve Bayes from Scratch (Kaggle Q4)** | Training Accuracy | 96.2 |
| | Testing Accuracy | 78.93 |
| Bernoulli Naïve Bayes | Training Accuracy | 92.48 |
| | Testing Accuracy | 76.2 |
| **Final Ensemble Algorithm** | Training Accuracy | 98.96 |
| | Testing Accuracy | 80.3 |

**Distribution Choice for Naïve Bayes**: I personally used three models of NB in my final ensemble algorithm, and to find optimal values for Alpha, I used the Grid Search Algorithm. My final Alpha value for the MultiNB model was around 0.15 (and Bernoulli at 0.10), which seemed optimal based on my hyperparameter search. We know that Multinomial NB makes the assumption that the data is multinomially distributed.

**Weighted Hard Voting in the Ensemble Algorithm**: I realized that when I fit the data individually for all the models, some models were more effective than others. Some overfit (eg: MLP and Adaboost), and some underfit (e.g.: Gaussian NB). I tested a number of voting combination in hard voting and soft voting, and hard voting seemed to work better for me, which is why I sticked to it for my submission[4].

**Regularization:** My Logistic Regression Model had the penalty set to 'L2' (SKLearn default)[5]. I noticed that the model with L2 performed better than an L1 or ElasticNet penalty. My MLP Model's penalty value alpha was increased a little bit too to improve overfitting (to 0.12).

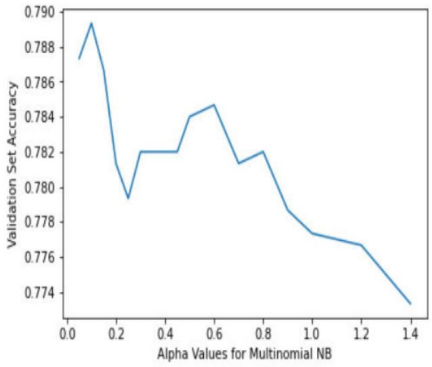
**Activation Function:** In my initial LSTM tests, I had tried both the Softmax Activation Function and the Sigmoid Activation Function, and Sigmoid seemed to perform better.

## Results:

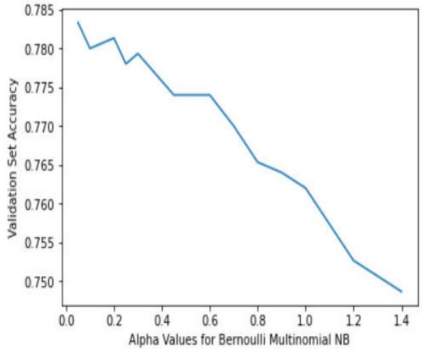
I had tested the following classical Machine Learning Algorithms for this project (given in the table to the top left).

From these classical Machine Learning Algorithms, I have selected the ones that not only perform the best, but have a reasonable bias-variance tradeoff. We have also tuned the hyper-parameter alpha (which is the additive smoothing parameter) for Multinomial Naïve Bayes, for the Bernoulli Naïve Bayes Algorithm and for the Gaussian Naïve Bayes Algorithm. Furthermore, I will also show how results vary when I change the Activation Functions for my LSTM RNN, and switch between different optimizers in Keras. First, let us show how the validation accuracy for Multinomial Naïve Bayes and Bernoulli Naïve Bayes varies with alpha (Grid Search):

```
Out[109]: [<matplotlib.lines.Line2D at 0x1b08460fdc0>]
```




```
Out[112]: [<matplotlib.lines.Line2D at 0x1b0846f1370>]
```



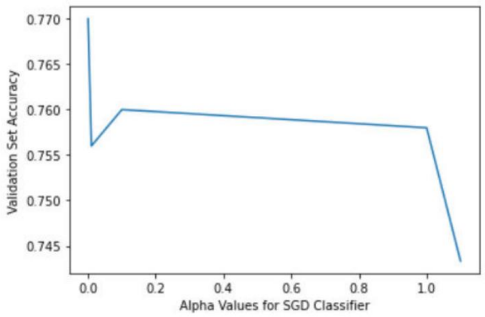

From the above curves, we can see that the validation accuracies are the highest for Multinomial NB for the alpha/smoothing value of 0.15, and the highest for Bernoulli NB for the alpha or smoothing value of 0.10. Hence, we have selected 0.15 and 0.10 for both the chosen algorithms respectively.

Furthermore, we also played around with the LSTM RNN configurations (which we tried for our first test), and we observed the following:

1. Softmax Activation Function, Adam Optimizer, Vocab Size – 30000, Sparse Categorical Crossentropy Loss, Epochs: 25 – Training, 0.972, Validation: 0.61
2. Sigmoid Activation Function, Adam Optimizer, Vocab Size – 10000, Sparse Categorical Crossentropy Loss, Epochs: 25 – Training, 0.993, Validation: 0.58
3. Sigmoid Activation Function, Adam Optimizer, Vocab Size – 500, Sparse Categorical Crossentropy Loss, Epochs: 25 – Training, 0.983, Validation: 0.662

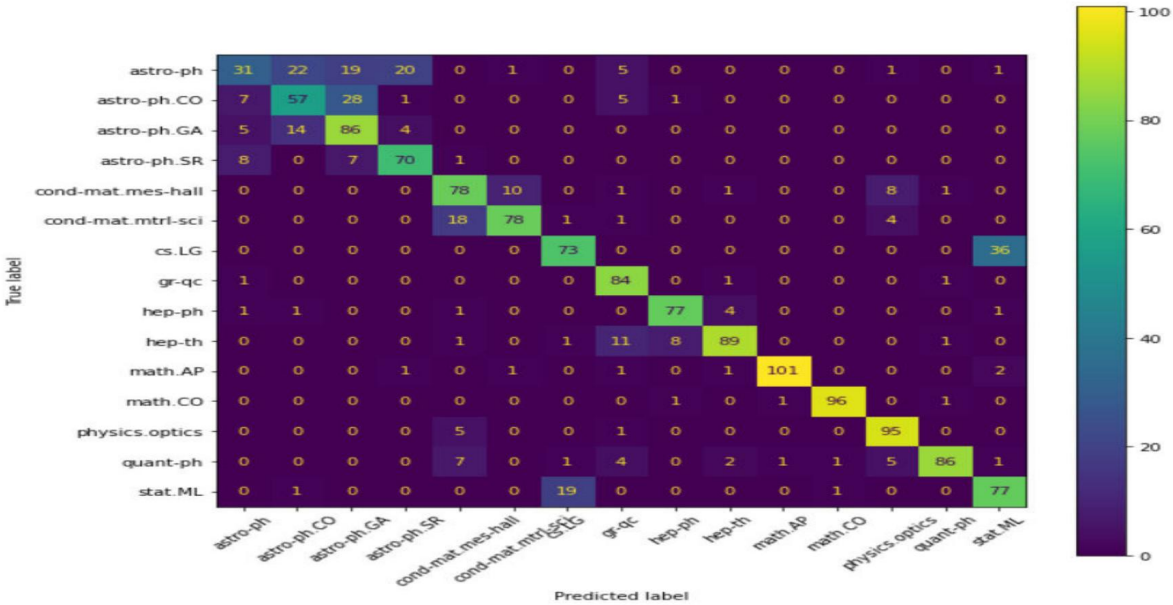
I have also played around with the Alpha (Learning Rate) values for the SGD Classifier. Here are my observations:

```
]: [<matplotlib.lines.Line2D at 0x1aeed43f340>]
```




I have selected the Alpha (Learning Rate) = 0.0001 value for this classifier for best results.

For the Naïve Bayes from Scratch component, we do not have any specific hyperparameter set, and we have not done any grid search. We obtain very similar accuracies as SKLearn's Naïve Bayes output (only ~1% difference overall on validation data).
Furthermore, we have also plotted the Confusion Matrix of our final ensemble method algorithm.




For our final model, we find out that the **F1 Score is 0.7911**, the **precision score is 0.7977**, and the **recall score is 0.79902.** Furthermore, I had also done two tests – pre-processing without Lemmatization, and pre-processing with Lemmatization. I had observed that Lemmatization indeed helps our ensemble algorithm overall, even though it seemed to negative affect some algorithms when they were trained individually.

## Overall Discussion:

In our project, we started out by testing many algorithms in an ad-hoc fashion on the given arxiv dataset, and then we selected the best pre-processing techniques, and then the best algorithms with a good bias-variance trade-off. Then, we ensembled them using SKLearn's Hard Voting Classifier with appropriate weights. I have disregarded algorithms which are too computationally expensive or have a terrible validation accuracy.
I believe there are several ways to ameliorate the overfitting that I faced in this project. The first way: Doing a more extensive hyperparameter search for all the algorithms used, and doing an exhaustive hyperparameter search for even the weights used in the Voting Classifier. I could not do an extremely thorough hyperparameter search as running algorithms like the MLP or *ExtraTreeClassifier* were very computationally extensive.
Secondly, I think Bagging could have helped our algorithms here, while improving both the training accuracy and the validation accuracy. I could not employ Bagging for our Final Algorithm due to a huge computation time.

**Why did I not get a better rank in the leaderboard despite having a stable algorithm?**
I have done the mistake of not running test.csv with a model fitted from the entire dataset, but it was fitted using 70% of the data (during the Holdout Cross Validation testing). We know that more data is the best regularizer, and this would have given us better results. I figured this mistake out very late.
**I have also noticed that the class "astro-ph" was often misclassified into other labels, as many physics based labels seemed to be present in other classes too. (e.g.: astro-ph.CO has a lot of similar words, as highlighted in the word cloud in the appendix).** Both of these classes do not have impressive accuracies.
Finally, I believe the project could have been made even better if I had tried out techniques such as Word2Vec and Doc2Vec for data vectorization.
We could also try out Transformer based Algorithms such as BERT and observe the results, since they have been seen to do well in the recent years (e.g.: GPT-3).

## Statement of Contribution:

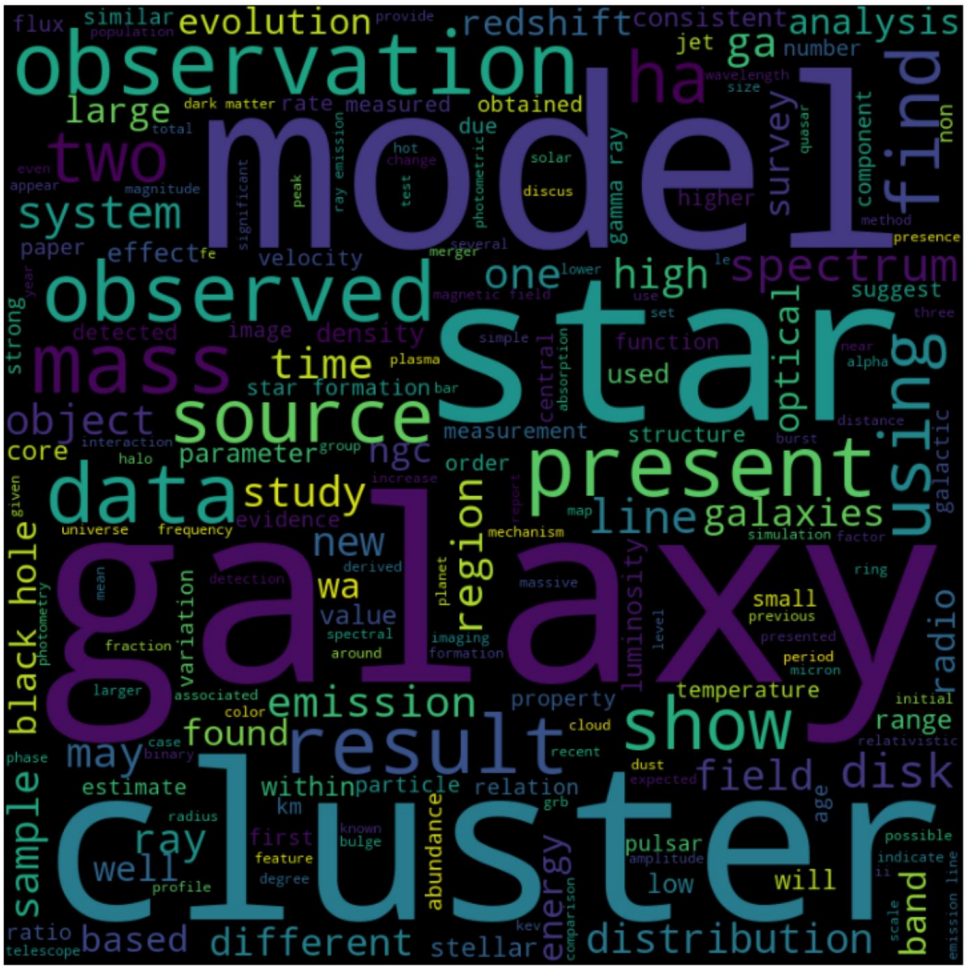
I hereby state that all the work presented in this report is that of the author. All research or blog papers used is referenced. I have done **ALL** work on my own.

**References:**

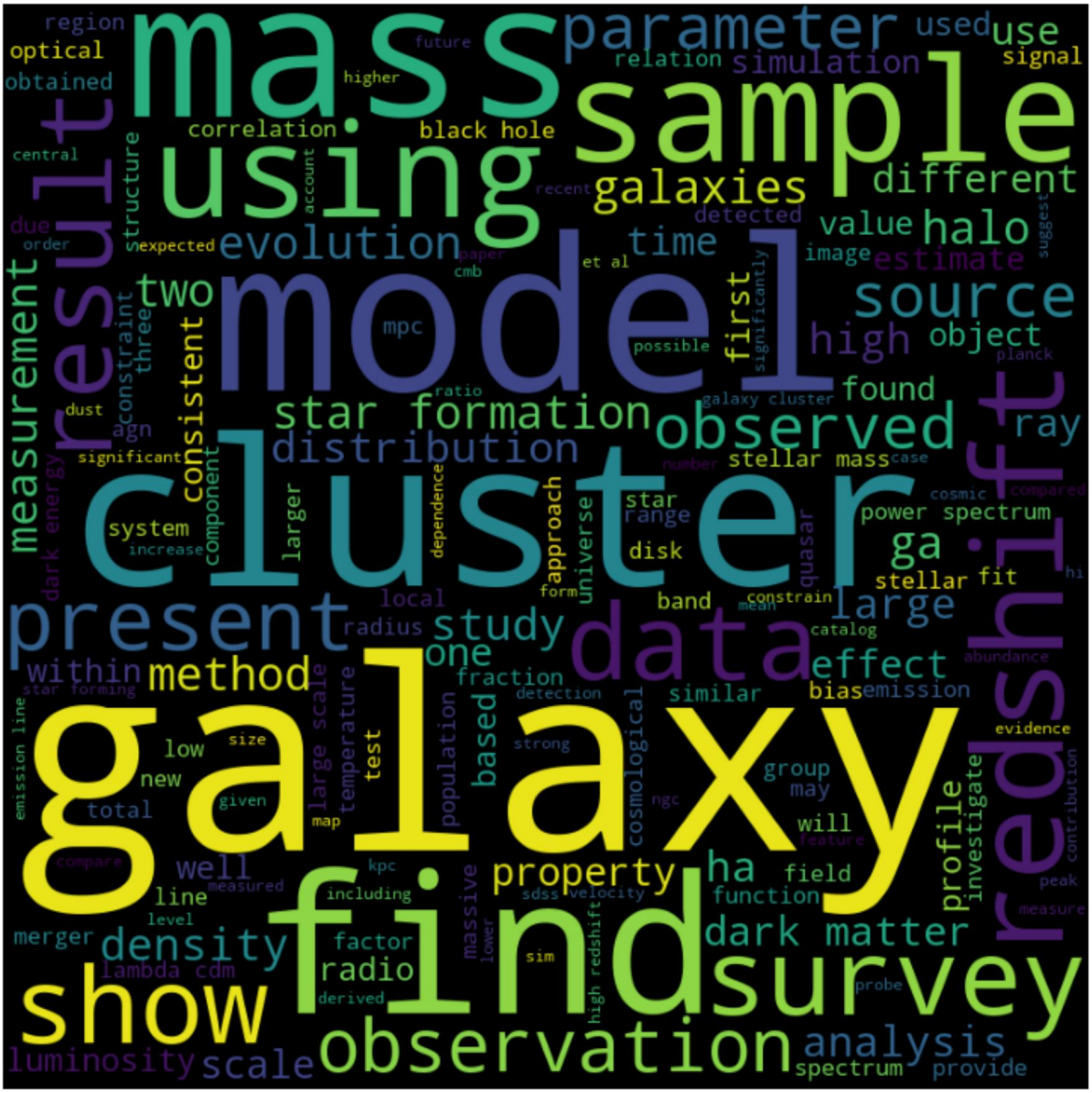

1. https://www.quora.com/What-is-the-difference-between-TfidfVectorizer-and-CountVectorizer-1#:~:text=CountVectorizer%20counts%20the%20word%20frequencies,document%20frequency%20(IDF)%20part.&text=CountVectorizer%20counts%20the%20word%20frequencies.,-Simple%20as%20that
2. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html#:~:text=An%20AdaBoost%20%5B1%5D%20classifier%20is,focus%20more%20on%20difficult%20cases.
3. https://medium.com/datadriveninvestor/k-fold-and-other-cross-validation-techniques-6c03a2563f1e#:~:text=Cross%2Dvalidation%20is%20a%20statistical,subsets%20from%20the%20same%20data.
4. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
5. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
6. https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd17


**Appendix:**

**astro-ph word cloud:**



**astro-ph.CO word cloud:**




**Note: We observe that the above word clouds share a lot of similarities. Probably the reason why both classes do not have extremely high accuracies.**

## 2.5 Discussion of results 4 / 4

✓ + **4 pts** **clear discussion of results**

+ **1 pts** superficial discussion of results

+ **0 pts** no discussion of results



gradescope