Text Classification using Machine Learning

Pulkit Khandelwal
260717316
pulkit.khandelwal@mail.mcgill.ca
Team: LazyWorkers

Seby Jacob
260656219
seby.jacob@mail.mcgill.ca
Team: LazyWorkers

Krtin Kumar
260713550
krtin.kumar@mail.mcgill.ca
Team: LazyWorkers

***Abstract*— This report studies the effectiveness of different machine learning algorithms on a text classification problem. Logistic regression, Multinomial Naive Bayes, Support Vector Machines, Random Forests and K-Nearest Neighbors are considered. The best model is chosen for each type of algorithm and an ensemble of these models is also considered. This study is performed towards Project-2 of Applied Machine Learning at McGill University in Fall 2016.**

I. Introduction

Classification problems are an integral part of machine learning. A very important subset of classification is text classification. While natural language programming and machine learning are pushing their frontiers in practical applications, we present a study on text classification into four categories, namely, mathematics, statistics, computer science and physics. Uni-grams and bi-grams are used as features after a chi-square correlation ranking. Logistic regression, Naive Bayes and Support Vector Machines are used to experiment, validate and make predictions on a test set using the proposed features. K-nearest neighbors and random forests were also implemented, but the complexity of the programs and their unrealistic run-times led to their dismissal.

II. Related work

Text classification is a complex problem to optimize, as there are various feature engineering and feature reduction methods available. Instead of using cross-validation for each and every task, we decided to use the knowledge from the existing work done in these fields. Below are some of the choices we made using previous studies.

II-A. *Feature Selection (Chi2, PCA, I-Gain)*

Feature selection helps us to reduce the dimensionality of the data set, by throwing out irrelevant and noisy features, while keeping only those which have a good correlation with the output labels or explains most of the variance. Feature selection is domain specific and depends on the kind of data one has. Three algorithms have been used in this study. Manning et al[5], suggests using Mutual Information to measure how much information is conveyed by the absence or presence of a particular feature in making the correct classification. Pearson's chi-squared [6] test gives an estimate of the dependence between a specific term and a specific class. Thirdly, the most widely used method in the Machine Learning community, Principal Component Analysis (PCA), brings out strong patterns in a data-set and reduces higher dimensional data into lower dimensions [7]. PCA did not work for KNN, hence was not used. In general, chi-square correlation gives a better estimate than Mutual Information, and therefore, the obvious choice for feature reduction in our implementation.

II-B. *Lemmatization vs Stemming*

Lemmatization and stemming, both are used for dimension reduction in text classification. Lemmatization reduces the word to its root word by removing only the inflectional part. This is the word which you use to look in the dictionary. On the other hand, stemming reduces the word to its stem, using predefined rules which vary depending upon the stemming algorithm used. Stemming may also result in removal of derivational morpheme. As shown in [10], lemmatization performs slightly better but at a large cost of speed. This improvement is more prominent in language modeling, where we don't want to loose information by removing the derivational morpheme. In our problem both should work equally good. We decided to use lemmatization, as we only had to process the text only once,freeing speed from being a constraint.

II-C. *Uni-gram and Bi-gram*

The most definitive method of feature representation for a text classification problem is the Bag of Words (BoW) approach. The use of uni-grams and n-grams(most frequently, bi-grams) have been studied extensively in literature. Caropreso et al[1], study the quality of n-grams on text classification using the Reuters data-set. After using mutual information to retrieve the highest scoring features, their experiments reveal that bi-grams are generally more useful in text classification than uni-grams. Bekkerman and Allan[3], conclude that while bi-grams are unlikely to provide an improvement in classification performance on unrestricted classification problems, their use in limited lexicon domains with stable phrase constructions can be fairly useful. Tan et al[2], finds bi-grams from uni-gram seeds with a minimum document frequency in at least one of the target categories. They are then sorted using highest occurrences and information gain before being incorporated into the training set. These experiments show significant improvement in classification performance on the "Yahoo!"Science data-set. Thus, based on the work done in the above cited papers, we decided to use bi-grams and uni-grams in our feature set, with a document frequency threshold.

III. Problem representation

III-A. *Text Preprocessing*

The raw text was first converted to lowercase, as case sensitivity is not required for classification. As explained in the previous section, we then used lemmatization; by first finding part of speech tags using NLTK Part of Speech tagger [11]. After this, NLTK Lemmatizer [12] was used and the output was stored in a text file. This process was done prior to Feature extraction and was done only once. During the above process the part of speech tags for each word was stored in a separate file.

III-B. *Choice of Features*

There are broadly two choices of features we could consider. One is the word counts or n-grams and the second is count of parts of speech tags. Upon validating the performance of parts of speech tags along with uni-grams and bi-grams, an increase in accuracy was not detected. Hence, parts of speech tags were eliminated from the feature set and only n-grams (uni-grams and bi-grams) were chosen as described in the following section.

III-C. *Feature Extraction*

For feature extraction, NLTK's TfidfVectorizer was used [13]. The TfidfVectorizer counts each word or n-grams in the document and returns a feature vector. The word counts are normalized by the frequency of that word in the entire selection of documents. It also generates a vocabulary on the training data, which is later used for testing. Some of the parameters that can be configured, are the following:
min_df: This is the minimum document frequency a word should have in order to be considered as a feature. This can either be an absolute number or a percentage of documents. We used min_df as **0.01 %** of the total examples (documents).
N-grams: We can configure TfidfVectorizer to use a range of N-grams. Using insights from the feature development studies in past literature, we use uni-gram and bi-gram counts as features. We consider that since the target categories are domains with high probability of strong phrase structuring, the use of uni-grams and bi-grams should give us the best possible results for classification in the proposed classes.
Stop words: The TfidfVectorizer also has a provision for excluding stop words. We excluded the stop words in the NLTK stop words corpus [14] and English punctuation.

III-D. *Feature Reduction*

For feature reduction as explained in section II-A, we used Chi square test to rank all the extracted features. The features were ranked from the smallest p-value to the largest and stored in a file. Different number of features were then selected, with a top down approach and the optimum number of features were selected using cross-validation as explained in section V.

IV. Algorithms and selection

We tested SVM, Logistic Regression with L2 regularization, Naive Bayes with Lidstone/Laplace Smoothing [15], Random Forests and KNN. With the exception of KNN and Random Forests, we selected the number of features and hyper-parameters of all other methods via 5-fold cross validation. After implementing Random Forest and KNN, we realized they were computationally expensive and hence, very slow. So, for these algorithms, we will only report the best possible performance by reducing the number of features and training examples, which were chosen based upon what was feasible for our machines. For all the algorithms written by us, we used one vs all method to account for multiple classes.

IV-A. *Logistic Regression*

We used Logistic Regression with regularization hyper-parameter λ. In particular we used *L2* regularization and tested over a range of values.

IV-B. *Naive Bayes*

We used a Multinomial distribution for Naive Bayes from sci-kit learn[15] with a smoothing parameter α. The smoothing parameter redistributes the weight of features, by transferring some of the weight to the features which have zero weight. α = 0 is no smoothing, α = 1 is Laplace smoothing and for any other α, it is called Lidstone smoothing.

IV-C. *Linear Hard SVM*

For implementing SVM we used CVXOPT Quadratic Programming solver [16]. For this we represented the equation in the dual form solution given in class notes, in a matrix form, which is compatible with the package.

IV-D. *Ensemble*

We made a custom model by using the predictions from the above models, namely Logistic Regression, Naive Bayes and SVM.We will refer to these as base models from now on. Along with this, we created four conflict resolution models. These models were basically created by choosing parameters for the base models, that maximized the precision for each category. Thus, for each category cs, math, stat and physics, different parameters were chosen based upon the performance on precision.In the end, we have a conflict solver for each category as well as the base models. If all base models had the same prediction, then the output value is unchanged. If there is a conflict in the base model prediction, then the conflict solver's prediction is trusted only if the prediction of the conflict solver matched with its category. In case no conflict solver matches,the prediction from our best base model is chosen. The intuition behind using conflict solver is that, if we train a model, specific to the highest precision of a certain category, the model certainty is maximized when a prediction is made to that category. A drawback of maximizing precision is that, it may lead to decrease in recall. Thus, the number of values in which the conflict resolver category and predicted category is same, will reduce by an amount (1-r_i)*(Number True prediction with class i), where r_i is the recall of conflict resolver for i$^{\text{th}}$ category.

IV-E. K Nearest Neighbor

Distance: Three distance metrics were used, namely, Euclidean, Minkowski and Radial Basis Function (RBF: a Gaussian kernel). The standard deviation (sigma) is chosen such that most of the data lies within 95 % of the kernel.

Nearest Neighbors (k): To choose the number of nearest neighbors, one can simply take the square root of the number of training examples [8]. Here, k can be taken as 291 (Square root of $\tilde{8}5$,000). To avoid the cases where a tie can occur, k should be chosen as even number when number of classes is odd and vice verse. Since, number of classes is 4 (even number) , we choose k to be an odd number.

IV-F. Random Forest

Classification of text into four categories were considered as four binary classification problems to develop random forests

Modified Entropy: An idea of modified entropy was used to grow trees as presented by Johnson et al[4]. For each feature *f*, a split value *v* is considered. *T* is the full range of examples, *T1* is the group $f \leq v$ and *T2* is the group $f > v$. Let $p_1 = P(y=1|x_i \in T1)$, and $p_2 = P(y=1|x_i \in T2)$, while $p = P(x_i \in T1|x_i \in T)$. A probability estimate transformation is now done on p,given by r : [0,1] $\rightarrow$ [0,1].

$$r(p) = \begin{cases} \frac{1}{2}(1+\sqrt{2p-1}) \text{ if p>0.5} \\ \\ \frac{1}{2}(1+\sqrt{1-2p}) \text{ if p}\leq 0.5 \end{cases}$$

Now, the modified entropy for p is given by:

$$g(p) = -r(p)log(r(p)) - (1 - r(p))log(1 - r(p))$$

For all the splits given by varying f and v, we associate a cost function that measures the impurity of the split.

$$Q(f,v) = pg(p_1) + (1 - p)g(p_2)$$

Number of Trees: The number of trees in the forest, was fixed at $\sqrt{n}$ where n is the number of training examples, since it is a good approximation based on the size of the feature space.

However, the complexity of the implementation was too high ($\propto$ to (4 * ntrees * nfeatures * nexamples)), to efficiently test and validate on a comparable chunk of training data.

V. Testing and Validation

To select the best model, we first chose optimum parameters for Logistic Regression, SVM and Naive Bayes. This was done using 5-Fold Cross Validation and choosing the best value based upon the F1 scores. The training and validation error were, continuously monitored to check for over-fitting. For feature selection, we varied the number of features from 5,000 features to 60,000 with a step of 5,000 for all the algorithms.

V-A. Logistic Regression

For Logistic Regression, the optimum number of features was found to be 50000 as shown in Fig 2 and the best regularization factor was found to be α=1/4. The inverse of regularization was varied from 1 to 5 as shown in Fig 1. These parameters obtained an accuracy of 92.774 % on validation set and 82.025 % on Kaggle Test set.

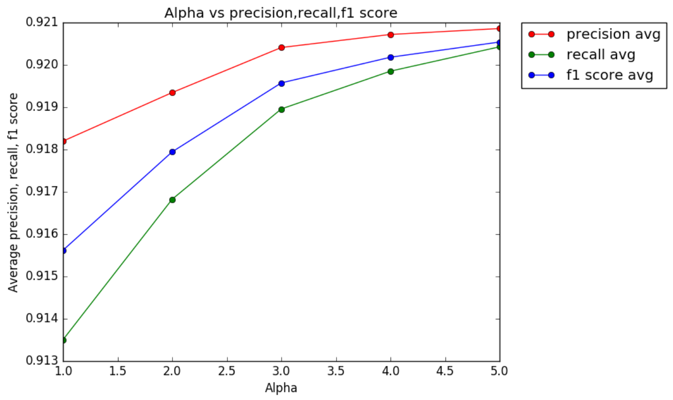


Fig. 1. Logistic Regression precision, recall, f1-score vs $\alpha = \frac{1}{\lambda}$

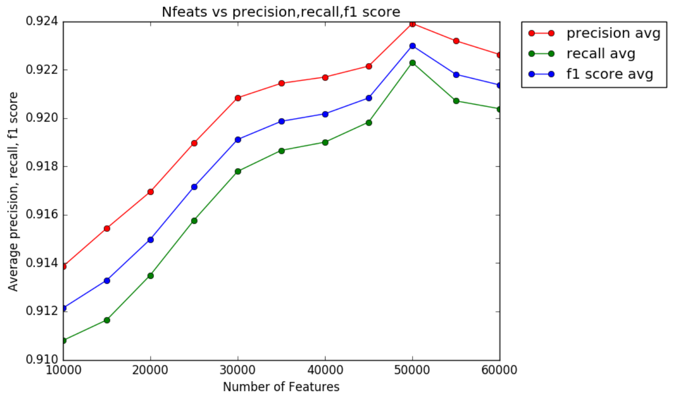


Fig. 2. Logistic Regression: $\alpha = 1/\lambda$, where λ is regularization factor

V-B. Linear Hard SVM

For SVM, the optimum number of features was found to be 40000 as shown in Fig 3, which gave an accuracy of 92.751 % on the validation set and 82.281 % on the Kaggle test set.

V-C. Naive Bayes Multinomial

For Naive Bayes with Multinomial distribution, the optimum number of features were found to be 55000 as shown in Fig 5 and the optimum smoothing value α was found to be 0.5. The smoothing parameter α was varied from 0.5 to 2.5 as shown in Fig 4. Using these parameters we obtained an accuracy of 91.29 % on validation set and 81.884 % on the Kaggle test set.

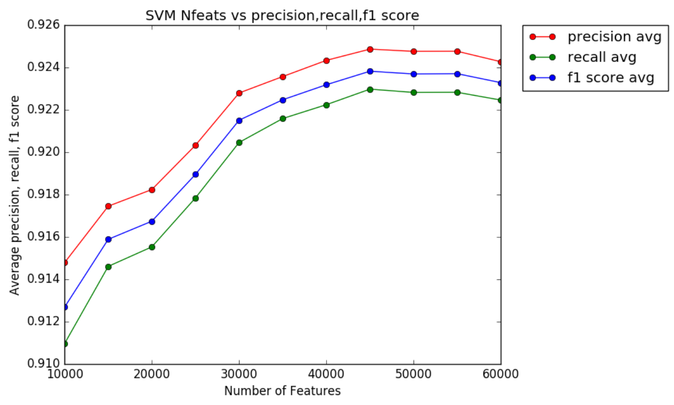


Fig. 3. SVM precision, recall, f1-score vs Number of Features

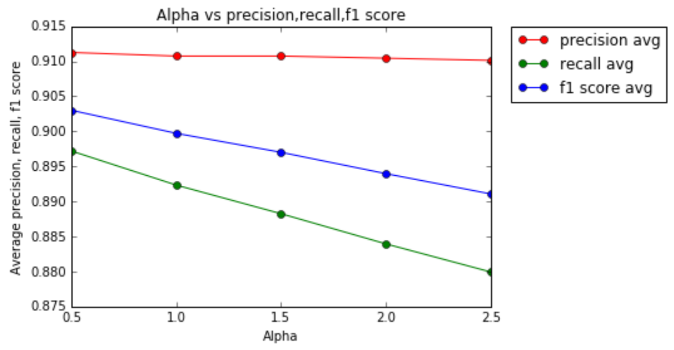


Fig. 4. Naive Bayes precision, recall, f1-score vs Alpha

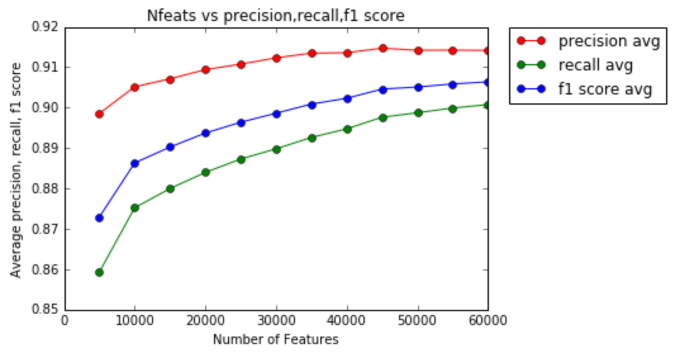


Fig. 5. Naive Bayes precision, recall, f1-score vs Number of Features

V-D. KNN

Due to computation speed constraints, we were not able to run KNN on the entire set of data. We ran it for 3000 features, 10000 training examples, with p=3 and nearest neighbors k=5 for Euclidean distances.

V-E. Random Forest

We ran Random Forests for 10000 examples and 5000 features, with number of trees as 10 and random feature range from 2 to 10.

Model	Train E	Valid E	Recall	Precision	F1	Kaggle
Logistic	2.71 %	7.23 %	92.47 %	92.55 %	92.5 %	82.025 %
SVM	3.51 %	7.24 %	92.45 %	92.58 %	92.51 %	82.281 %
Naive Bayes	6.6 %	8.7 %	90.75 %	91.45 %	91.06 %	81.884 %
Ensemble	-	-	-	-	-	82.294 %
Random Forest	18.15 %	20.55 %	79.13 %	78.34 %	78.73 %	-
KNN	21.56 %	22.1 %	76.86 %	78.4 %	77.45 %	-

TABLE I
RESULT SUMMARY

The best results for all the models[1] are shown in Table I.

VI. DISCUSSION

Best Performance: Overall, the best performing method, based on the test results in Kaggle, is the Ensemble method as described in section IV-D (also shown in Table I). This shows that a combination of majority voting and conflict solvers enhanced the predictions. Only a minor improvement is observed,because we were very strict in resolving conflicts. However, having a strict conflict resolution also ensures that the performance of the ensemble, will always remain within a small positive δ of our best solo algorithm.

Category Analysis: On analyzing how the top 3 algorithms have performed under each category,(the category-wise results are shown in Table II),we observe that Naive Bayes has been outperformed by Logistic Regression and SVM in all the four categories on F1 score. However, Naive Bayes has better precision for Statistics and better recall for Computer Science. It was able to do so, only by compromising on recall and precision respectively as informed by the low F1 scores. Comparing F1 score for Logistic Regression and SVM, we observe that Logistic Regression performs better only for predicting Computer Science correctly.For all other categories, SVM has an upper hand. Further, we observe that Statistics is the category which is the hardest to predict and Mathematics is the easiest, as seen in Table II. The performance can be enhanced in all areas by using information from external sources or better feature selection methods like PCA, which we were unable to do due to a huge data-set and low computational resources.

SVM vs Logistic Regression: An interesting point to note here is that, even though Logistic Regression performs better in terms of validation error by 0.01 %, SVM is better in terms of the F1 score. When testing both these models on the actual Kaggle test set, it is clear that SVM is the better model between the two.

VII. STATEMENT OF CONTRIBUTIONS

- **Defining the problem:** Done as a team.
- **Developing the Methodology:** Done as a team.
- **Performing the Data Analysis:** Done as a team
- **Coding the Solution:**
  KNN - implemented by Pulkit
  Random Forest - implemented by Seby
  Logistic Regression and SVM - implemented by Krtin

[1]Download the detailed results of all models, from this link.These results were used to find the best performing model

Model	math	cs	stat	physics
Precision SVM	97.01 %	90.74 %	88.88 %	93.72 %
Precision Logistic	96.95 %	91.14 %	88.43 %	93.71 %
Precision Naive Bayes	95.82 %	87.38 %	89.54 %	93.04 %
Recall SVM	94.71 %	93.06 %	89.43 %	92.61 %
Recall Logistic	94.77 %	93.06 %	89.61 %	92.48 %
Recall Naive Bayes	94.08 %	92.63 %	86.71 %	89.58 %
F1 SVM	95.85 %	91.88 %	89.15 %	93.16 %
F1 Logistic	95.85 %	92.09 %	89.02 %	93.09 %
F1 Naive Bayes	94.95 %	89.93 %	88.1 %	91.27 %

TABLE II

CATEGORY WISE PRECISION, RECALL AND F1 COMPARISON

- **Writing the Report:**
  KNN and Logistic Regression - Pulkit
  Random Forest and Naive Bayes - Seby
  Logistic, SVM and Ensemble - Krtin

  ”We hereby state that all the work presented in this report is that of the authors unless otherwise referenced”

REFERENCES

[1] Caropreso, Maria Fernanda, Stan Matwin, and Fabrizio Sebastiani. .^{A} learner-independent evaluation of the usefulness of statistical phrases for automated text categorization.”Text databases and document management: Theory and practice (2001): 78-102.

[2] Tan, Chade-Meng, Yuan-Fang Wang, and Chan-Do Lee. ”The use of bigrams to enhance text categorization.Ïnformation processing & management 38.4 (2002): 529-546.

[3] Bekkerman, Ron, and James Allan. Using bigrams in text categorization. Technical Report IR-408, Center of Intelligent Information Retrieval, UMass Amherst, 2004.

[4] Johnson, David E., et al. .^{A} decision-tree-based symbolic rule induction system for text categorization.ÏBM Systems Journal 41.3 (2002): 428-437.

[5] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schtze, Introduction to Information Retrieval, Cambridge University Press. 2008.

[6] Chernoff, Herman, and E. L. Lehmann. ”The use of maximum likelihood estimates in chi^2 tests for goodness of fit.”The Annals of Mathematical Statistics 25.3 (1954): 579-586. APA

[7] Ding, Chris, and Xiaofeng He. ”K-means clustering via principal component analysis.”Proceedings of the twenty-first international conference on Machine learning. ACM, 2004.

[8] http://stackoverflow.com/questions/11568897/value-of-k-in-k-nearest-neighbour-algorithm

[9] Fernndez-Delgado, Manuel, et al. ”Do we need hundreds of classifiers to solve real world classification problems.”J. Mach. Learn. Res 15.1 (2014): 3133-3181.

[10] Vimala Balakrishnan and Ethel Lloyd-Yemoh ”Stemming and Lemmatization: A Comparison of Retrieval Performances ”

[11] http://www.nltk.org/book/ch05.html

[12] http://www.nltk.org/_modules/nltk/stem/wordnet.html

[13] sklearn.feature_extraction.text.TfidfVectorizer

[14] http://www.nltk.org/howto/corpus.html

[15] http://scikit-learn.org/stable/modules/naive_bayes.html

[16] http://cvxopt.org/

Appendix

Fig. 6. Logistic Regression F1 Score Plot C=0.5

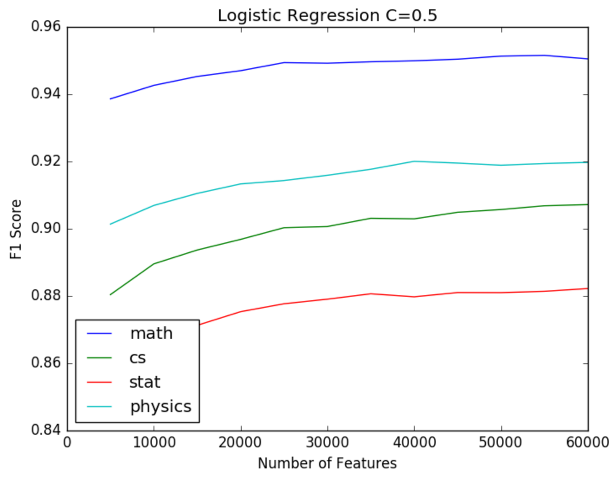


Fig. 7. Logistic Regression F1 Score Plot C=1

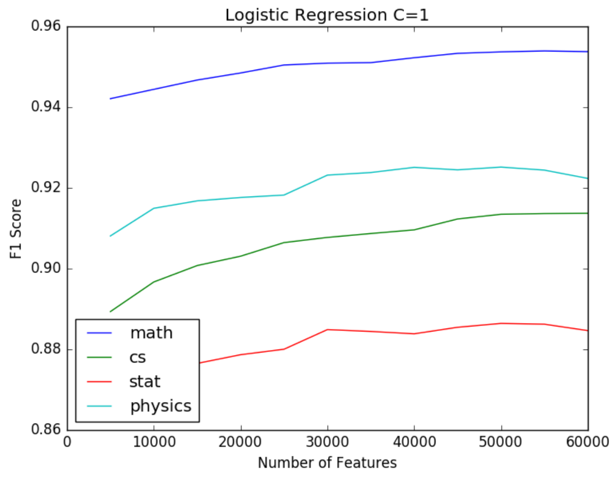


Fig. 8. Logistic Regression F1 Score Plot C=2

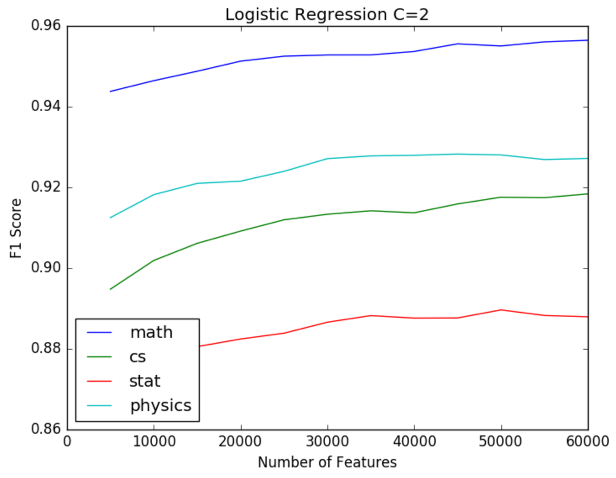


Fig. 9. Logistic Regression F1 Score Plot C=3

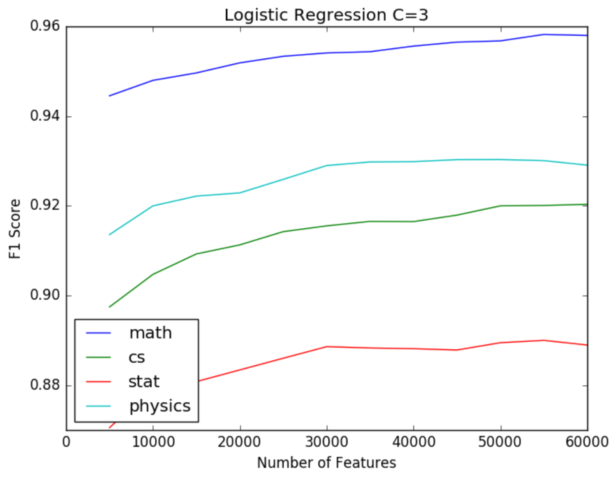


Fig. 10. Logistic Regression F1 Score Plot C=4

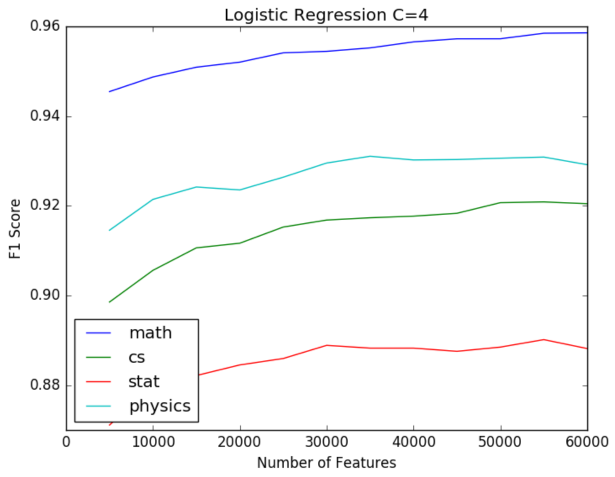


Fig. 11. Logistic Regression F1 Score Plot C=5

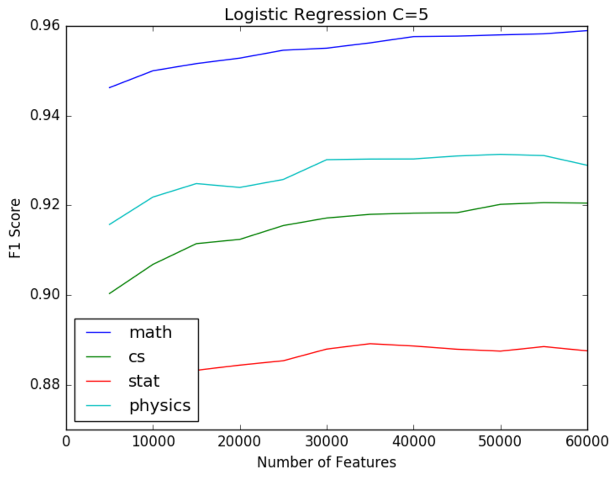


Fig. 12. Logistic Regression Precision vs Recall Plot C=0.5

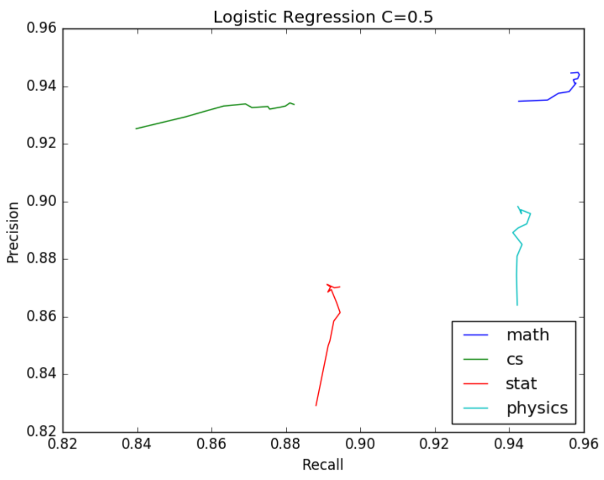


Fig. 13. Logistic Regression Precision vs Recall Plot C=1

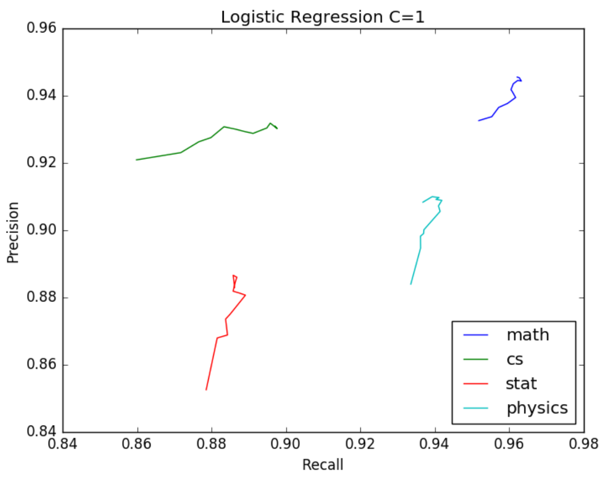


Fig. 14. Logistic Regression Precision vs Recall Plot C=2

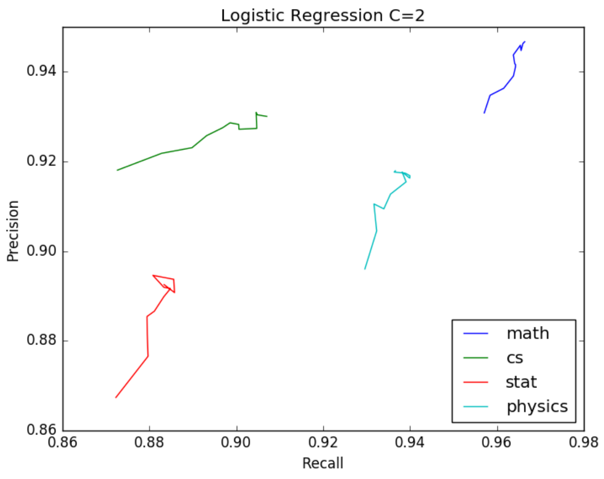


Fig. 15. Logistic Regression Precision vs Recall Plot C=3

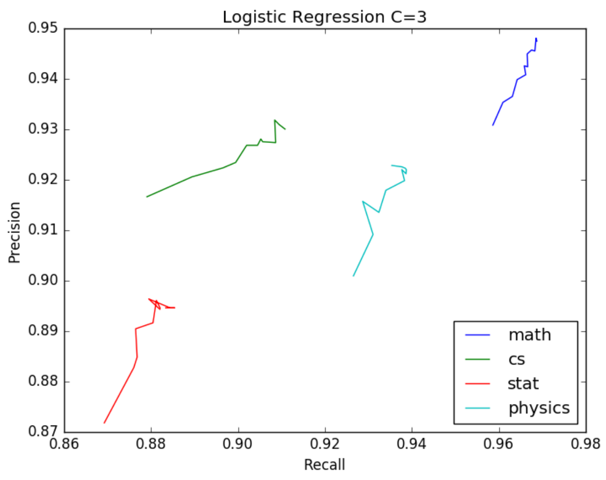


Fig. 16. Logistic Regression Precision vs Recall Plot C=4

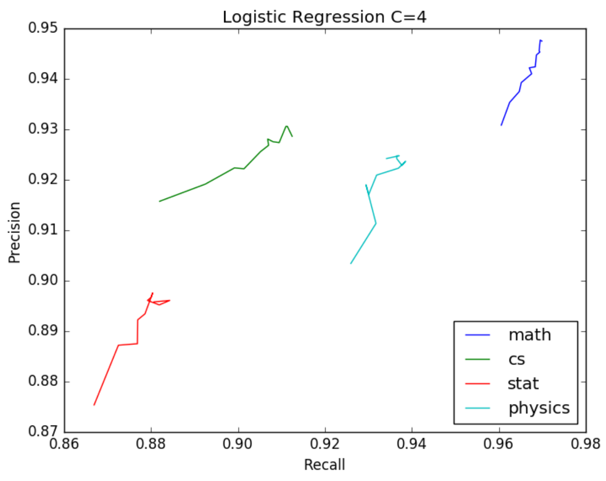


Fig. 17. Logistic Regression Precision vs Recall Plot C=5

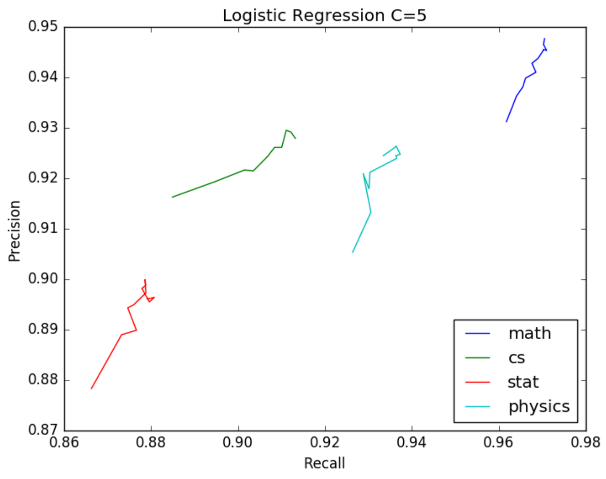


Fig. 18. Logistic Regression Error vs Number of Features Plot C=0.5

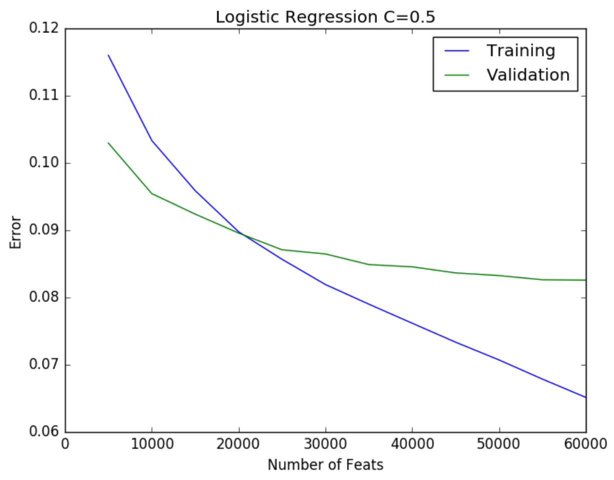


Fig. 19. Logistic Regression Error vs Number of Features Plot C=1

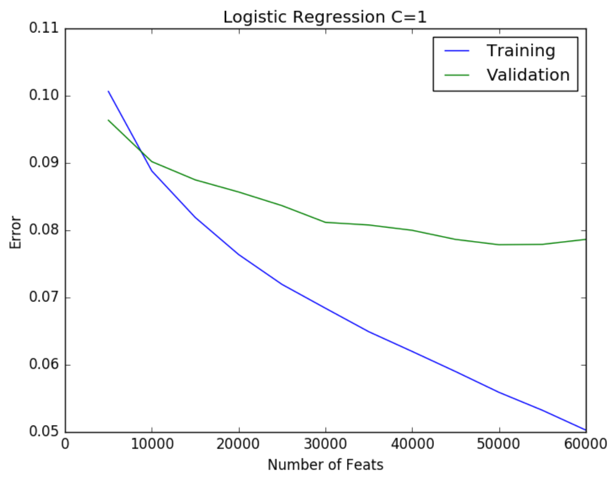


Fig. 20. Logistic Regression Error vs Number of Features Plot C=2

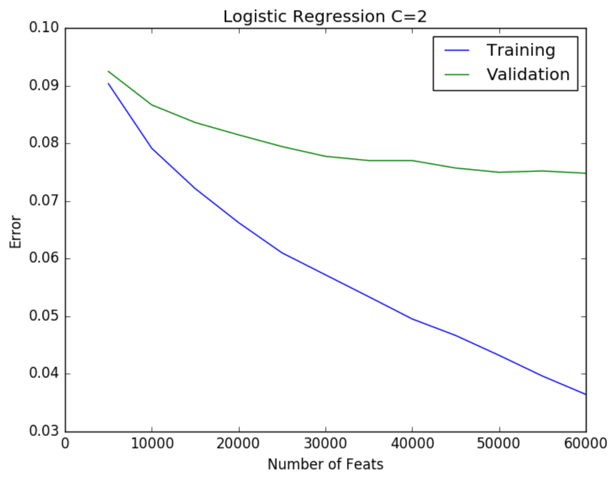


Fig. 21. Logistic Regression Error vs Number of Features Plot C=3

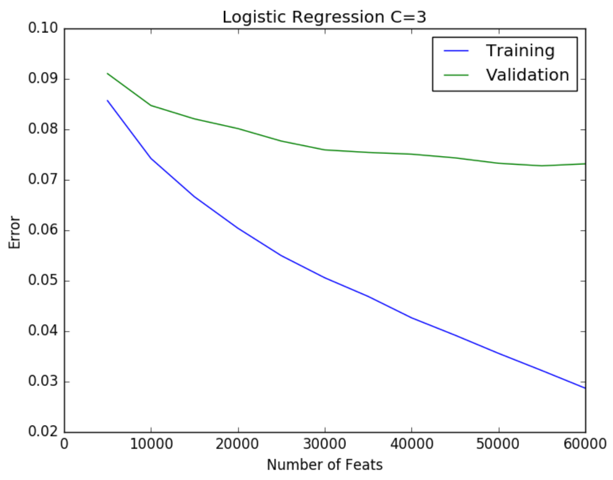


Fig. 22. Logistic Regression Error vs Number of Features Plot C=4

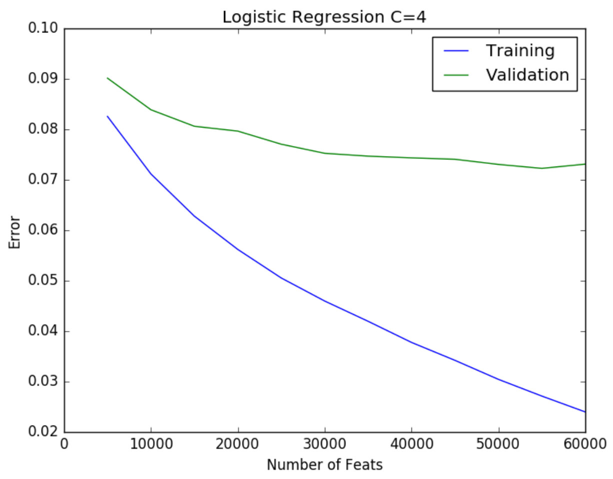


Fig. 23. Logistic Regression Error vs Number of Features Plot C=5

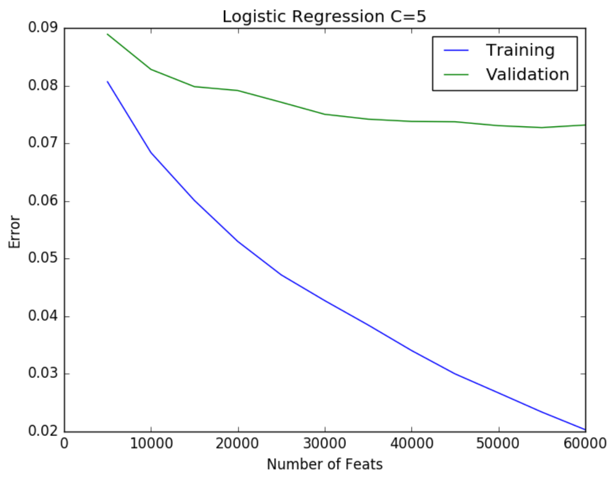


Fig. 24. Naive Bayes F1 Plot C=0.5

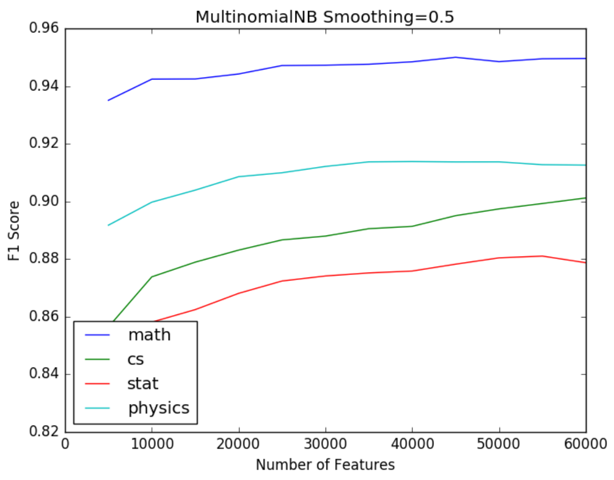


Fig. 25. Naive Bayes F1 Plot C=1

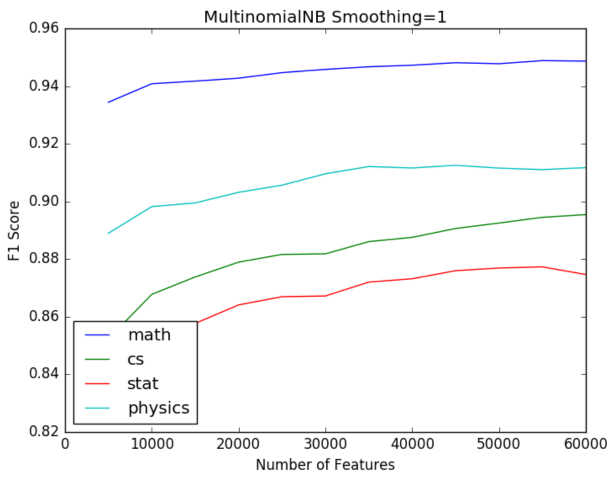


Fig. 26. Naive Bayes F1 Plot C=1.5

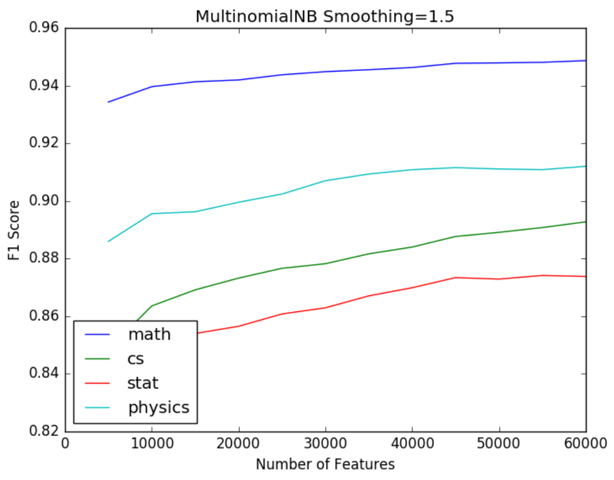


Fig. 27. Naive Bayes F1 Plot C=2

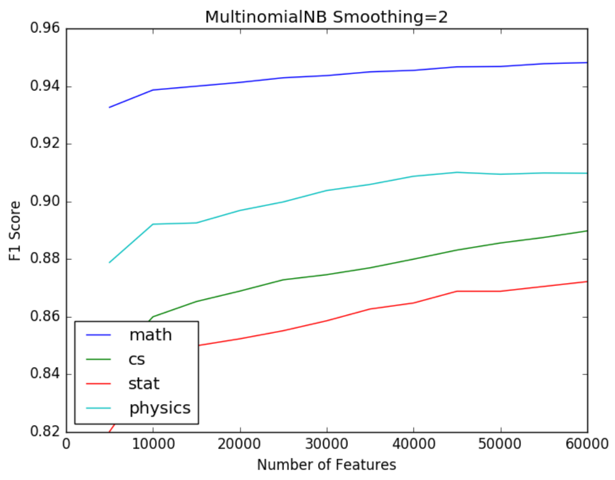


Fig. 28. Naive Bayes F1 Plot C=2.5

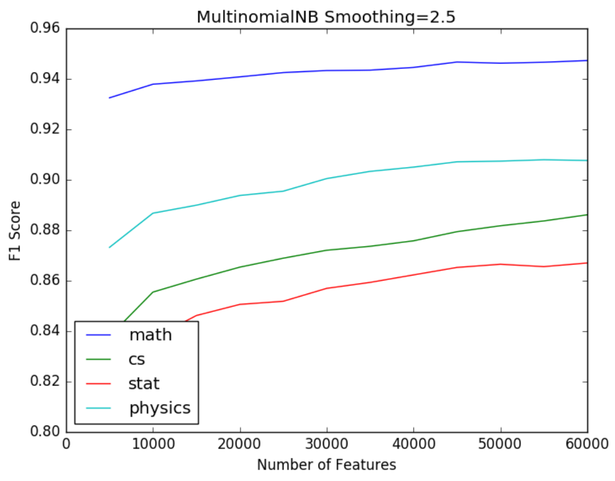


Fig. 29. Naive Bayes Precision vs Recall C=0.5

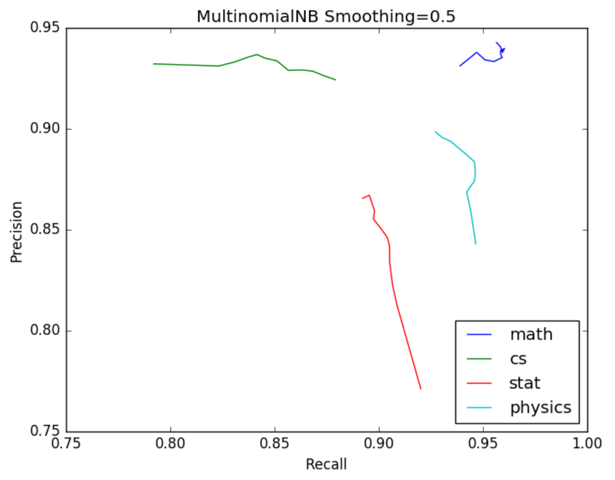


Fig. 30. Naive Bayes Precision vs Recall C=1

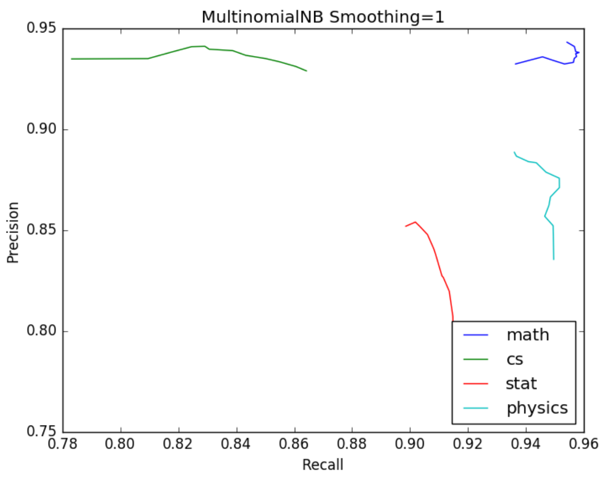


Fig. 31. Naive Bayes Precision vs Recall C=1.5

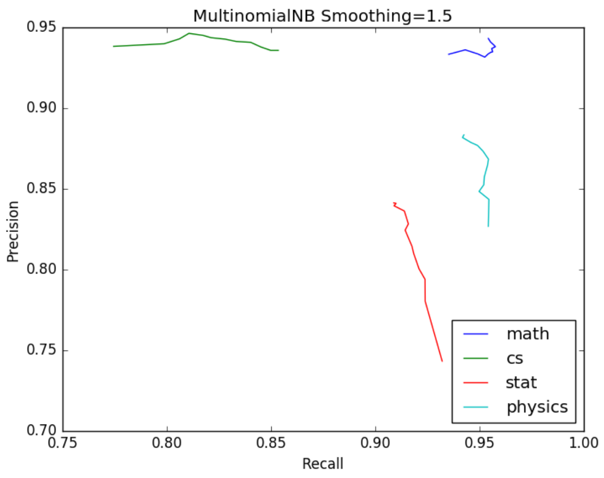


Fig. 32. Naive Bayes Precision vs Recall C=2

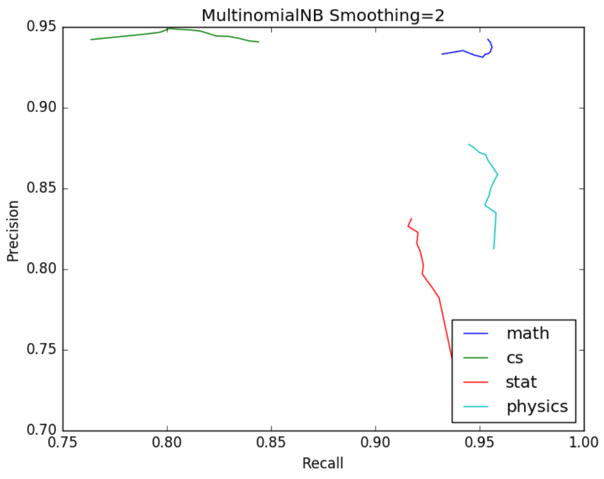


Fig. 33. Naive Bayes Precision vs Recall C=2.5

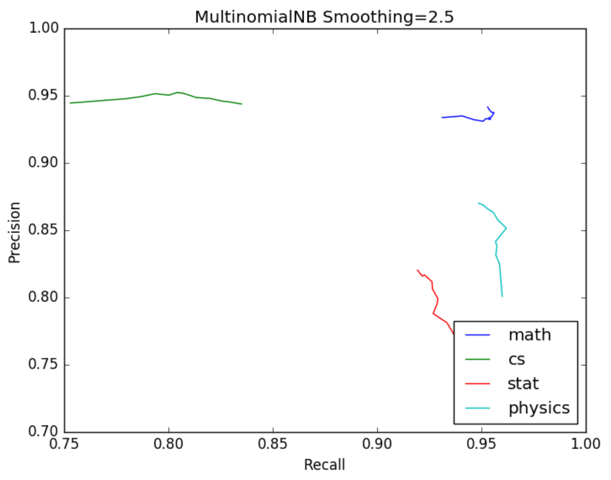


Fig. 34. Naive Bayes Error vs Number of Features C=0.5

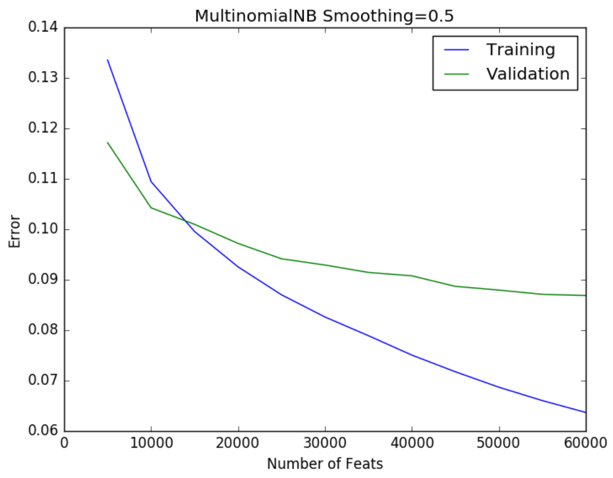


Fig. 35. Naive Bayes Error vs Number of Features C=1

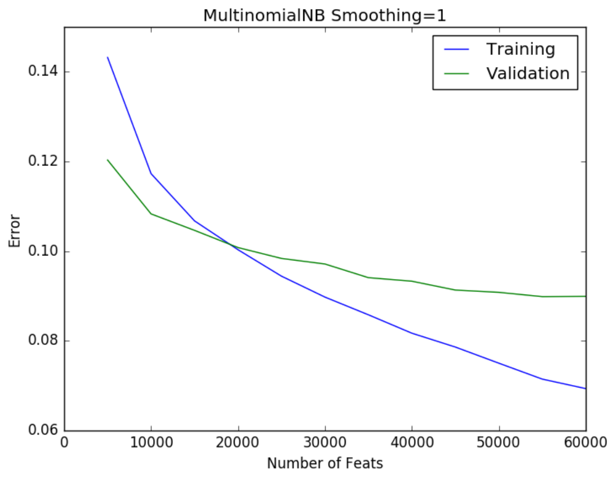


Fig. 36. Naive Bayes Error vs Number of Features C=1.5

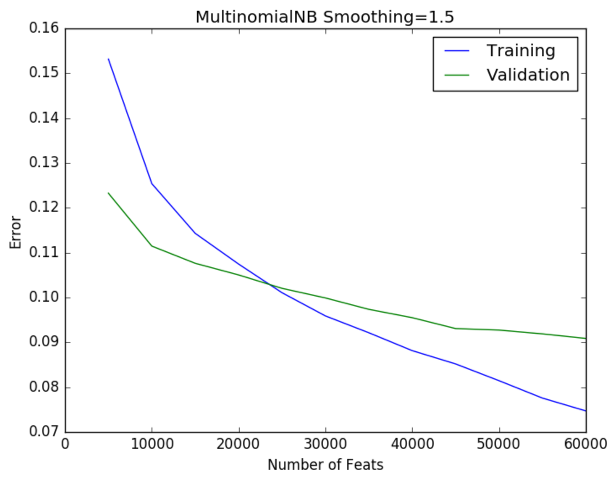


Fig. 37. Naive Bayes Error vs Number of Features C=2

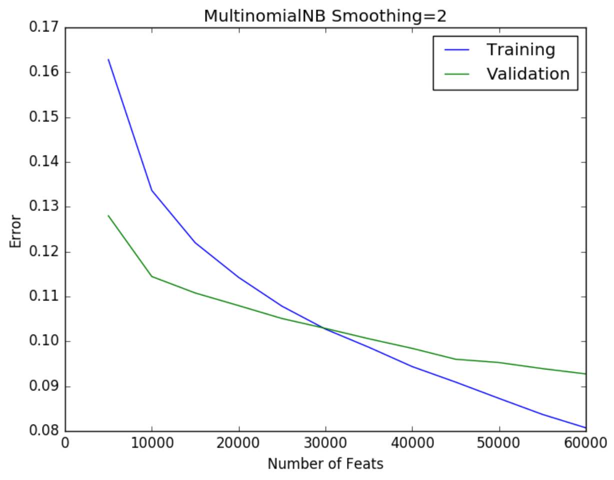
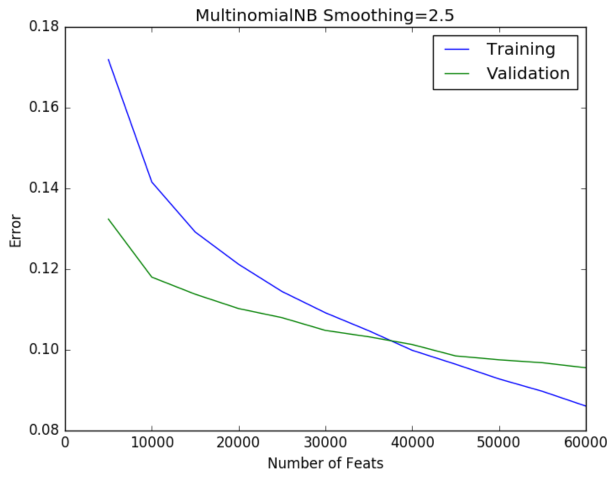


Fig. 38. Naive Bayes Error vs Number of Features C=2.5


MultinomialNB Smoothing=2.5
Training
Validation
Error
Number of Feats