ICSI499 Capstone Project Report

Evaluating Machine Learning Models for Product Classification

Project Team

Pranjal Atrey (001334399)
Alex Kisiel (001106797)
Jordan Le (001417470)
Erik O'Hara (001351890)
....
College of Engineering and Applied Sciences
University at Albany, SUNY

Project Sponsor

Dr. Brian Ruttenberg
Google
6425 Penn Ave, Pittsburgh, PA

12-1-2020

Acknowledgements

We thank our sponsor, Dr. Brian Ruttenberg, for his ongoing support and guidance throughout the project; he has provided invaluable insight and knowledge towards our research.

We thank our professor, Dr. Petko Bogdanov, for his indispensable feedback on our milestones and accomplishments, as well as guidance throughout the project.

We thank the data.world community for providing public access to the datasets used in our work.

Abstract

Automatic product classification has become an important aspect for retail companies as online retail is quickly becoming the dominant method of shopping. This is a challenging task due to the large number of product categories and inconsistent and inaccurate classifiers. Several e-commerce datasets are used to evaluate, compare and optimize four particular machine learning methods in classifying products, with a focus on model robustness, accuracy and the effect of the data sets on the models themselves. The four methods are as follows: [1] support vector machine (SVM) classifier, [2] logistic regression classifier, [3] naive bayes classifier, and [4] decision tree classifier.

From our results, we conclude that [1] smaller datasets tend to have lower accuracies, [2] classifiers that output better accuracies take more time and classifiers that take the least time output lower accuracies, [3] a classifier trained on one dataset may not work well for other datasets, and [4] model robustness depends on the noise (i.e. importance of words in the product names, etc) which can decrease or increase the accuracy and effect the trend.

Contents

1 Introduction

Online retail is quickly becoming the dominant method of shopping for many consumers, which puts pressure on sellers to format their listings as efficiently as possible. As such, their goal becomes categorizing items for sale for the purpose of consumers finding them quickly. For shops with relatively few items, this isn't too much of a problem. However, for larger operations it can prove costly, both in terms of time and manpower. The solution is to automate as much of the process as possible, while retaining an acceptable degree of classification accuracy. This can prove to be difficult when applying machine learning techniques for classification. The overarching goal in this project is to evaluate and optimize various machine learning methods for product classification, with a focus on model robustness, accuracy and the affect of different data sets on the models themselves.

The most basic solution for classifying products has been manually creating a number of predetermined categories and slotting items in. While this method boasts high accuracy, its major drawback is the cost of time and consistent supervision. Existing solutions also examine machine learning models, such as the naive bayes classifier (based on bayes' theorem) [5] – a probabilistic approach – to determine whether an item is likely to fit in a category. This method can be difficult to implement, however, since it requires a prior knowledge of each dataset, which we might not have when training the program on something it's never seen before. In addition, it also has shown to be less accurate than other methods tested since it assumes statistical independence between the features, which is almost never true in practice. One additional method [2] takes a middle road between the previous two: the program tries to probabilistically ascertain the product's category, and determines a numerical measure of confidence for its guess. The programmer then sets a 'confidence threshold,' allowing the program to pass on the product to human eyes for manual classification if the minimum confidence level is not met. The major drawback for this method is that it does not guarantee completely automatic classification.

The core idea is to assist users to find the best classifier method possible, weighing both accuracy and time spent against each other. A classifier is 'efficient' when both its accuracy and time have equal relevancy in the program; put simply, it should be fast enough to run within a reasonable amount of time, while being accurate enough to classify correctly at an acceptable rate. In addition to the existing solutions, we also focus on impact of different kinds of noise in our dataset and how that can impact the accuracies. Below are the main goals of research:

1. Classifiers should work on any dataset, given that it conforms to a predetermined format
2. Classifiers should be efficient, fast, and accurate to a certain degree
3. Classifiers should be compared and contrasted against each other in addition to different datasets

The project consists of researching four different machine learning classification methods and testing how effective they are in terms of accuracy and time, implementing optimization techniques to achieve better results, and creating a simple user-interface for users to classify products.

2 Background and Related Work

There has been an extensive history of research for classification methods and machine learning, specifically with consumer product classification. Due to the rapid rise of technology over the years, along with the proliferation of e-commerce stores such as Amazon and Walmart, shop owners must have automated systems that do the heavy lifting of classification for them. Amazon has created a method for their users to upload a bulk of inventory items by using a spreadsheet in CSV format [1]. Ease of management has been accomplished using this technique; however, users are required to enter all information of their product manually, which is a tedious task. Accuracy will obviously be high due to human intervention, but efficiency and speed will drop drastically.

Recently, there has been research conducted on implementing machine learning into product classification [5]. Bayes' Theorem – a probabilistic approach – is used to determine whether an item is likely to fit in a category. Furthermore, text processing features such as stemming, number removal and word removal have also been used. It also has shown to be less accurate than other methods tested since it assumes statistical independence between the features, which is almost never true in practice. Such work can verify that our results are innovative and correct. In addition, though the same text processing features are not used in our study, this provides a great example of how noise in the data can effect the results.

One additional method [2] takes a middle road between the previous two: the program tries to probabilistically ascertain the product's category, and determines a numerical measure of confidence for its guess [2]. The programmer then sets a 'confidence threshold,' allowing the program to pass on the product to human eyes for manual classification if the minimum confidence level is not met. Though it is beneficial to have high accuracy, the major drawback for this method is that it does not guarantee completely automatic classification.

Another approach [3] has used a multi-level taxonomy tree. A product's natural language description is translated into a sequence of tokens representing a root-to-leaf path in a product taxonomy. The approach demonstrates that the machine translation models shown in the paper can propose meaningful new paths between previously unconnected nodes in a taxonomy tree, transforming the taxonomy into a directed acyclic graph (DAG). This approach is valuable in understanding how to handle translated datasets, and how tokens and taxonomy trees are used in the study.

Compared to the research mentioned above, the novelty of our work lies in the comparison of different datasets, in addition to word embedding techniques to test for model robustness. One of our datasets is originally in French, but has been translated for comparison purposes.

3 Proposed System/Application

3.1 Overview

Our system/applications looks to solve the issues that were previously discussed. Nicknamed 'classi.py', our program and research aims to cover the holes that have been left in previous solutions to the problem of efficient online product classification. To assist the shop owners with classifying, we have created a user interface that will allow them to input any dataset and train a classifier using that dataset and one of four methods - Naive Bayes, Logistic Regression, Decision Tree, or SVM. After the classifier has been trained, the user can then classify as many products as they'd like, including features such as product name, product color, or price. Based off the dataset that the classifier was trained on, it will give its best guess as to which category the product belongs in. Given the database, the categories it will try to predict will be the categories that already belong inside of the initial dataset. For example, if you train a classifier with nine categories, it will predict into one of those nine categories and none else. This is what is shown to the end-user to make their experience as efficient and seamless as possible.

$$AVG = \frac{\Sigma(array)}{array\_length} \tag{1}$$

For data analysis, we have multiple scripts set up that the front-end users do not interact with at any stage. To calculate average classifier accuracies (discussed in future Tables 3 and 4), we devised a script that trained the classifier n=20 times, adding its reported accuracy to an array with each passthrough. After 20 iterations, we calculated the average using Equation 1. In addition, we calculated the standard deviation (i.e. the amount of dispersion between data points) using Equation 2 and the variance (i.e. the spread of numbers between data points) using Equation 3.

$$\sigma = \sqrt{\frac{\Sigma|x - \bar{x}|^2}{n}} \tag{2}$$

$$\sigma^2 = \frac{\Sigma|x-\bar{x}|^2}{n} \tag{3}$$

A separate script allows us to run trials on the future Table 5. This script trains a specific type of classifier using one dataset, and runs testing products from another dataset through the newly trained classifier. Using this method, we train and test on two separate datasets using categories that are found within both datasets. As a simple example, both the NewChic and Walmart datasets have a 'Beauty' category; this script trains a classifier on one of these datasets, finds all of the products that are in the same category in the other dataset, and determines whether the classifier can accurately predict and place them back in the same category.

Lastly, one final script was written to create plots and graphs for display of our data. Because some of the datasets were quite large – and some of the methods took a great deal of time to train – we used Python's 'pickle' utility to store several already-trained classifier objects created in classi.py in their own files. By doing this (and labelling the 'pickled classifiers' uniformly by their method and the size of training data withheld for their training), we were able to programmatically load them in and create graphs at a much faster rate. Each of the graphs shown were obtained by taking the accuracies/training times for each method at ten-percent intervals of training data withheld (i.e. 10%, 20%, 30%, ... ,90%).

3.2 System Requirements

3.2.1 User Classes

Our system and application has very specific use cases to it. For the program, there will be three different user classes - **Administrator, Users, and Customers. Administrators** are the top-level users of the program and will be those who have been granted access to modify the code. As time goes on, more efficient processes may become relevant, and thus will need to be implemented by these administrators. This user class will also be the one to make sure the program has the necessary libraries installed and to point out any prerequisites. The other user class is the **User** class. This latter class includes the shop owners that will be using the program to classify. These users will have the ability to input a dataset to train a classifier on as well as input their products to be automatically classified. They will also be responsible for having their datasets in a specific format that agrees with our program. Lastly, there is the **Customer** user case. The customer will have no access to the back-end of the project like the administrators and users, rather, they will be exposed to the end product of the system. After the Users run their dataset and classify their products through the classifier, the customers will see the result of these classifications, such as looking at tags or search results on Amazon or eBay. You may reference the use case diagram for this program found in Figure 1.

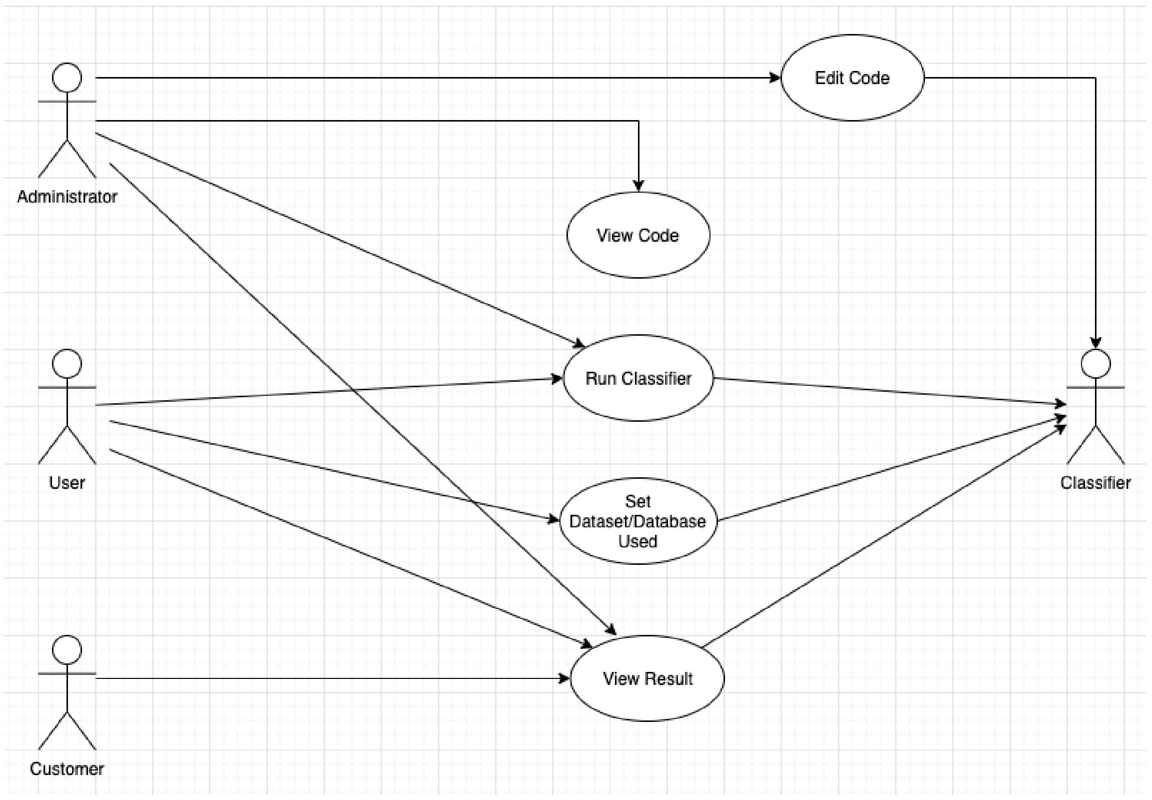


Figure 1: Use Case Diagram of classi.py [Source: https://bit.ly/2HY7jIE]

3.2.2 Functional Requirements

Now that the user classes are defined, we can also define their functional requirements. For **Administrators**, their functional requirements are to host and maintain the scripts; they thus ensure that no changes are required and no updates have been applied to the scripts. They will also be required to keep proper documentation. Documentation will be given for the base program, however, it will need to be modified and saved if they modify any part of the program to fit their needs (this includes updating the documentation as new updates are received). A **User** will need general adaptability skills, an understanding of how product classification works, and, optionally, how to find and filter products. For the adaptability, they will need to modify and adapt their dataset to the format that works with our program (i.e. the columns in the CSV file must match the format we have laid out). Finally, the **Customer** will have to know how to find and filter products on the website where the classifications will be applied. For example, the customers will have to understand how to do a specific search for the product and category that they are shopping for.

3.2.3 Non-Functional Requirements

For the non-functional requirements, there are three sub-categories - **performance, availability, and reliability.** For **Performance**, the project needs to be fairly efficient and run reasonably quickly (while maintaining a high degree of accuracy). Performance will vary depending on the user's machine, but the program's design should fundamentally reflect the need for efficiency (i.e. choice of algorithms, data parsing/tokenization/featurization, etc.). For **Availability**, the project and scripts should be freely available to anyone who would like to use it, hosted presumably on a third-party site like Github; the site will also provide documentation and code-use examples. Finally, for **Reliability**, the program needs to successfully – an reliably – accomplish its tasks. By nature, product classifiers rely on some degree of probability to function, which means it will guess incorrectly from time to time. The goal of this project is to minimize such errors while also fulfilling all of the above performance and usefulness requirements. Concerning the code itself, the publicly-available source should be updated as necessary to ensure that its dependencies are updated and everything runs smoothly.

3.2.4 Operating Requirements

For operating requirements, there is only one requirement that the user must specifically deal with: all scripts used, including the program itself, have been written in Python 3.6+. To use this product or any of its scripts, the user must make sure it has Python 3.6+ installed. Some features used, such as f-strings, have not been implemented before this version and as such will crash the program if attempted to open in a previous version.

3.2.5 Design & Implementation Constraints

There are multiple design and implementation constraints for the project to run as it was intended. Firstly, as described above, Python 3.6+ must be installed on the host machine where the program and/or scripts will be run from.

The program additionally uses certain Python libraries, several of which do not come pre-installed with Python (including, notably, scikit-learn). The Administrators and Users will therefore have to install these external libraries through a means such as 'pip' inside of a command line. We will include script files that will automatically install these libraries through 'pip' on multiple operating systems, but otherwise, the user needs to make sure they have all necessary libraries for this program. You may find all such libraries below in Section 3.4.

Further, the dataset will need to be in a proper format. It is required that the imported dataset is in CSV format and follows the format put forth in Table 1. The first column will need to delineate the product's category, the second column its subcategory, the third column its name, and the fourth column its price. Any additional information should be placed in columns 5 to column ∞; each such column will have its contents appended as an extra word to the products name for classification.

Table 1: CSV Format

Category	Subcategory	Name	Price	...
cat_1	subcat_1	product_name	product_price	...
cat_2	subcat_2	product_name	product_price	...
cat_3	subcat_3	product_name	product_price	...
...	...	...	...	...

Finally, the CSV tables should be in English only. The original NewChic dataset, for example, contained many items whose names and categories were in French; it was translated to English for ease of readability and use. The classifier may technically work for other languages, but the only officially supported language is English.

3.3 Technical Design

For the back-end and data analysis, we have set up multiple scripts that shall not be used by the end-user. For the main classi.py program, the SciKitLearn library is used to assist in vectorizing the passed datasets for the classifier's 'bag-of-words', as well as implementing each of the four classifier methods. The program is split into two separate functions, one to train a classifier and one to test data, named train() and classify() respectively.

train() takes an input file name, a classifier method, an optional testing data percentage value (representing the percentage of data that will be withheld from the training process for testing), and an optional parameter that adds a randomly-generated string of length n to the product's name. The input file – which must be in CSV format – will first be opened and parsed. The program assumes that each row of the file contains a product's category, subcategory, and name (in that order), then treats all additional features thereafter effectively as an extension of its name. This latter treatment is due to our classifiers' dependence on a 'bag-of-words' model, which entails assigning a value to each word in an array of words (in our case, the product's name/additional features), counting the number of times each word occurs in a given category (across many products for best results), and assigns values to each word based on how relevant the program perceives a word is to its category. Thus, if the program encounters a product whose name includes a particularly important word for many elements in a given category, it will know to put the product in that category. train() trains the classifier on every element of the dataset unless its 'train' parameter is set. If an integer is passed to this value, the method randomly selects that integer's percentage of the entire dataset and excludes it from the training process; this allows the program to input these elements as test data (which the classifier will not have seen) at a later point, and afford the user a measure of classification accuracy. The 'train' parameter also accepts a file name, which we used to ensure that each classification was trained on the exact same (randomly-selected) data elements. To do this we simply passed a file containing indices of data elements to withhold from training, and popped them from the training array to the testing array. The final, optional parameter allows the user to generate strings of random alphabetic characters of length n, and append them to each product's name; this was exclusively created and used for our fourth research question, examining the classifiers' robustness when a random string was added to their bag-of-words. train() returns the classifier and its various components as a dictionary which can then be used inside of the classify() method.

The classify() method is a good deal simpler than train(): the method accepts a list of products and the aforementioned dictionary full of classifier information. Within the dictionary is an actual classifier object – defined in the ScikitLearn library – which has a predict() method; classify() calls this methods on each element in the string array argument. The optional training process in train() does not use this classification method, but instead handles it itself. Finally, the program will print to the user the most likely category based on its prediction. You may reference Figure 2 to view a data flow diagram for the main classi.py script.

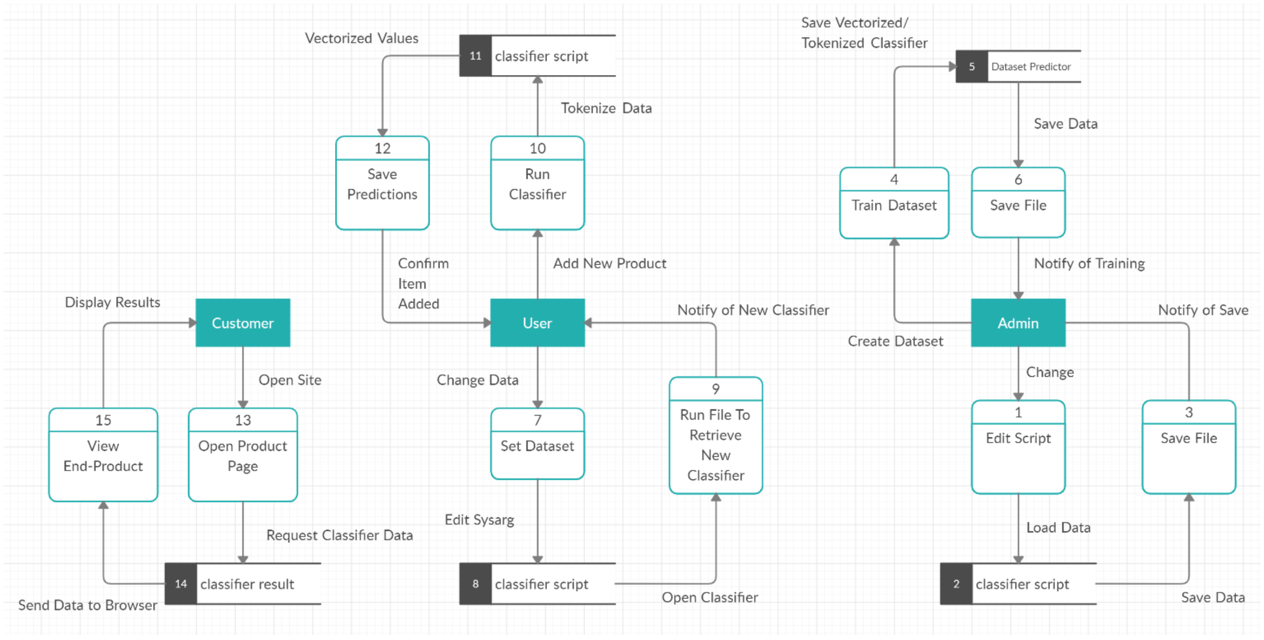


Figure 2: Data Flow Diagram of classi.py [Source: https://bit.ly/3ofbS0L]

For all testing scripts (such as the accuracy scripts and category accuracy scripts laid out in Section 3.1), the program will run the same script in the paragraph above, but will have extra features to it. Both of those scripts have a for-loop running for n times, collecting the accuracies of both and appending them into an array that holds the accuracies. After the for-loop ends, the average is calculated using Formula 1, and these values will be used later in the report for analysis.

3.4 System Implementation

The programming tools used in our study involve Python and Jupyter Notebook. Specifically, many Python libraries were implemented into the code, including:

- SciKitLearn [sklearn]

- Numpy
- Tkinter
- Random
- Pickle
- CSV
- SciPy
- MatPlotLib

In this project we used the following five **computer science theories**:

1. **Support Vector Machines.** A support vector machine (SVM) separates a given dataset to the best of its ability by using multiple hyperplanes to split the data and choosing the best hyperplane available. To find the optimal hyperplane, an SVM looks at the support vectors of the classes that are being compared, and uses the margin between these support vectors to determine if the hyperplane is satisfactory; the larger the vector margins between the classes the better. Smaller vector margins indicate a poor choice of hyperplane. Having a good hyperplane helps the classifier define which data belongs to a class, thus providing a more accurate result. [4]

2. **Logistic Regression.** Logistic regression (LR) is a binary classification method, in which a threshold is specified that indicates the value at which an element will be put into one class or another. As our study involves multi-class classification, the LR classifier is trained on just the examples belonging to one class vs. all the examples of all other classes. Similarly, all other classes are also independently classified against all others. After the classifiers have learned to distinguish their chosen class from other classes, the classifiers are run and the most confident LR classifier gets used.

3. **Term Frequency - Inverse Document Frequency (tf-idf).** TF-IDF is a technique used to quantify a word as it relates to its document as a whole. As mentioned in the explanation of the train() method, our programs implement a 'bag-of-words' method, which relies on quantifying each word based on how relevant it is to the category it belongs to. Through tf-idf, a weight is placed onto each word signifying its contextual importance within the document. The

result of this is that words important to each category are given a higher importance value than those that are comparatively less useful in this respect (words such as 'the', 'and' and 'of' are given lower priority, since they generally aren't category-specific).

4. **Naive Bayes.** A Naive Bayes classifier is based on conditional probability and uses Bayes' theorem, shown in Equation 4.

$$P(\theta|\mathbf{D}) = P(\theta)\frac{P(\mathbf{D}|\theta)}{P(\mathbf{D})} \tag{4}$$

   Using the Bayes theorem above, we can find the probability of θ happening, given the occurrence of D. For our purposes, the program attempts to probabilistically determine the likelihood of a product belonging to a category by looking at the tf-idf representation of a dataset's product names.

5. **Decision Tree.** A decision tree is built through binary recursive partitioning, which is an iterative process of splitting the data into partitions. We begin at the tree root where the data is split based on the feature that results in the largest information gain. As the process is iterative, the splitting procedure is used at each child node until the samples at each leaf node all belong to the same class.

We used the following three **software development fundamentals**:

1. **Code Reuse.** Code reuse is a software design practice where existing code is used for a new function or software. In other words, a new function is implemented using an existing software asset. In our work, we have implemented this practice in training and classify the datasets. We are able to use the same functions to classify all the datasets. Specifically, the training() and classify() methods are being reused within classi.py, as well as the utility methods for using pickle and generating graphs/tables

2. **Modular Design.** Modular design is a software design practice where the software itself is split up into smaller subparts that are easily modifiable and changeable. This allows for parts of the program to be changed without needing to replace the entire program. In this work, Classi.py utilizes this design by

initializing different methods which also allows for different user input. Specifically, the training() and classify() methods are separated to allow one or both of these functions to run at once. Both functions also allow for the input of different user variables such as train()'s classifier method, dataset, amount of testing data to withhold from the classifier training process, and string-addition size parameters.

3. **Regression Software Testing.** Regression testing verifies recent code changes do not alter the existing functionality of the code. In this project, we implemented regression testing to ensure the correct and complete use of our code. For instance, other classification methods in Classi.py were also checked while adding new classification methods to ensure results did not drastically change, ensuring that the code works as expected.

4 Experimental Design and Testing

4.1 Datasets

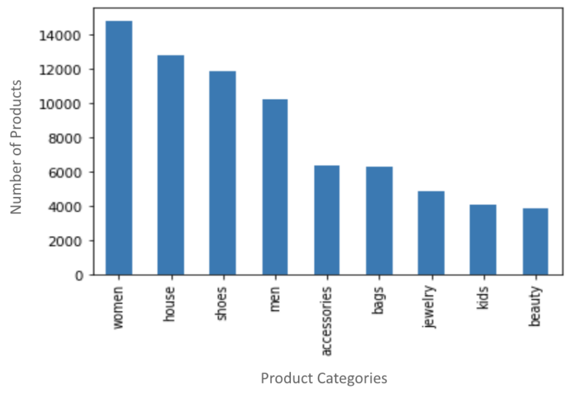
Three datasets are used in our experiment. Specifically, we examine data-sets from e-commerce retailers Walmart, Amazon and NewChic. These datasets are publicly available on the data.world website. Table 2 depicts the sizes of the datasets along with the number of categories for each.

Table 2: Dataset Sizes

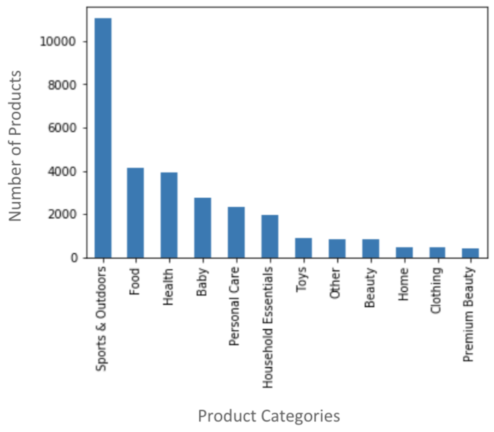
Data-set	Size (# of Products)	# of Categories
NewChic	75,000	9
Walmart	30,000	18
Amazon	10,000	26

Figure 3 depicts the number of products in each category for each dataset. Particularly in the Amazon and Walmart datasets, some categories have significantly more/fewer products than others, which can cause issues while training/testing on a model. A classifier trained on a dataset that is skewed toward one category in particular may disproportionately favor that category in its predictions, which would negatively affect its accuracy.

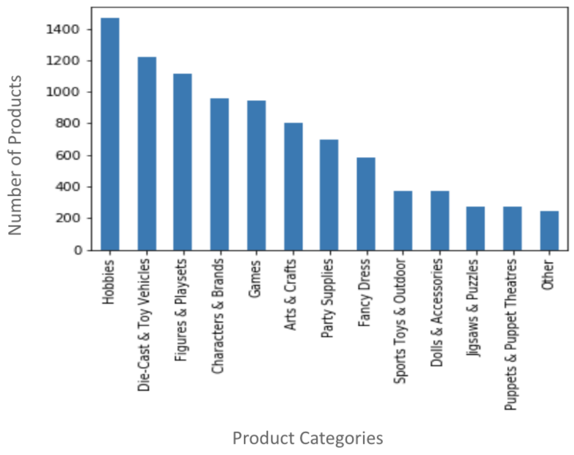
In order to remove the imbalance and limit its effect on the results, two solutions were implemented. Firstly, products in categories with less than 200 items were removed and added to a newly-created miscellaneous category called "Other". This method helps reduce the level of imbalance at the cost of some category variety: low-element categories are lost entirely, but there's less risk of misclassification into those categories. The other solution involves using class weights to weight the categories accordingly to their sizes. Class weights place more emphasis on the minority classes such that the classifier can learn equally from all classes. Since Naive Bayes is a generative model, balancing data isn't part of the true process; therefore, the method does not contain class weights.



(a) NewChic dataset



(b) Walmart dataset



(c) Amazon dataset

Figure 3: Number of Products in Each Category

A confusion matrix is an evaluation technique which helps in visualizing the weight of each class label in the dataset; Figure 4 shows the confusion matrix calculated for the NewChic dataset. Each row represents an actual category in the dataset, while each column represents the categories predicted by a classifier. The diagonal values in the matrix are very high, which indicates that each class is being weighted and labeled above 90%. Therefore, we can infer that the imbalance issue has been fixed.

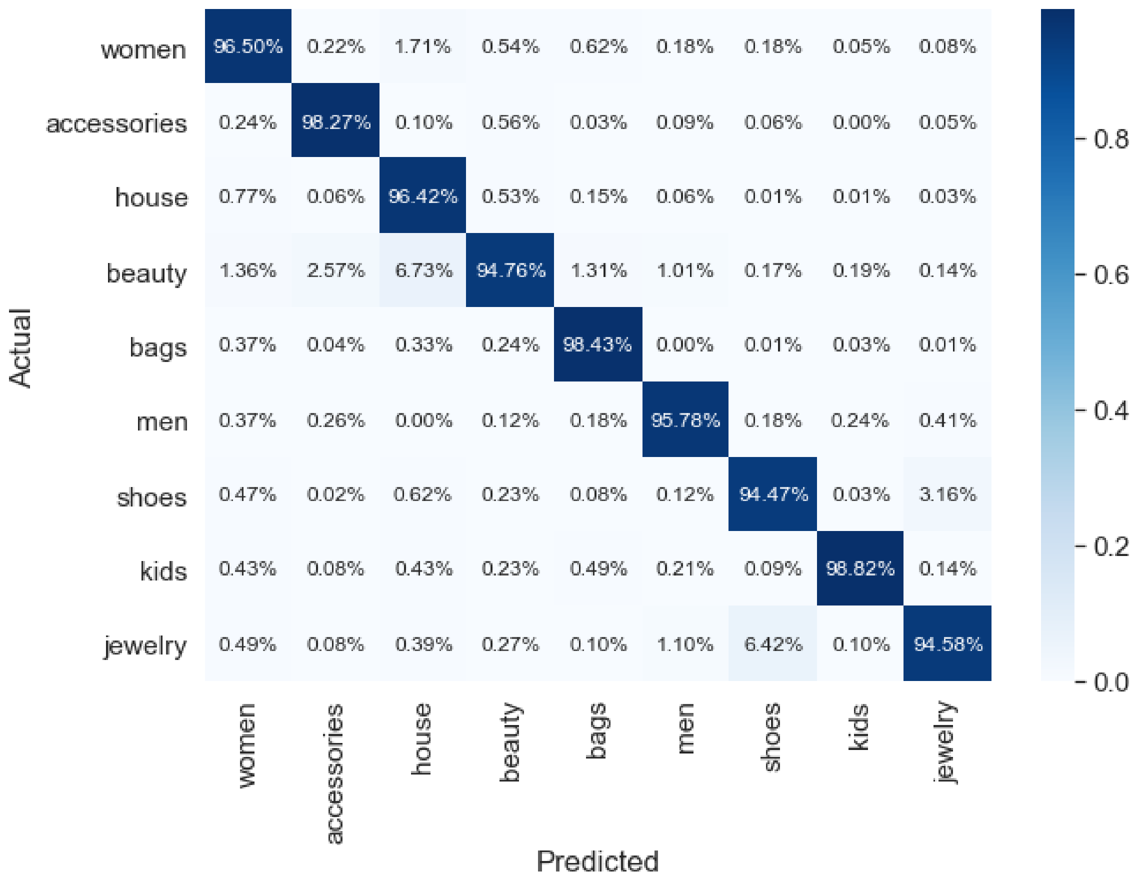


Figure 4: Confusion Matrix of NewChic Dataset Using Logistic Regression

4.2 Analysis & Results

Our study was based on the following four research questions:

1. How does the size of a dataset affect the accuracy of the classifier? (RQ1)
2. Which classification method is analytically better or worse? (RQ2)
3. How accurately does a classification method classify on datasets other than the trained dataset? (RQ3)
4. How robust are the classification methods? (RQ4)

4.2.1 How does the size of a dataset affect the accuracy of the classifier?

To understand the impact of dataset size on the accuracy of a classifier, we first used three different datasets - one from a French online e-commerce site called NewChic, and two more from e-tail giants Walmart and Amazon. As shown in Table 2, the datasets have 75,000, 30,000, and 10,000 products respectively. Table 3 displays the average accuracy for each classification method when run on all three datasets. The four different types of classifiers were trained and tested on the same dataset each time, with an average value being recorded over n=20 tests and 20% of the dataset being withheld for testing on the dataset.

Table 3: Dataset Classification Accuracies

	Naive Bayes	Logistic Regression	Decision Tree	SVM
NewChic	90.90%	94.00%	82.95%	94.00%
Walmart	76.15%	83.65%	71.65%	85.20%
Amazon	81.55%	86.90%	78.95%	87.20%
Averages over n=20 tests, Percent Testing Data Withheld = 20%				

These trials show that the largest dataset had the highest accuracies, while there was no correlation between the second and third datasets. Thus, we cannot conclude anything about the sizes of these datasets. However, we can run trials on the same dataset, splitting it into progressively smaller sizes. This will make sure the same products and categories are being used, rather than comparing them against abstract data.

As seen in Table 4, making sure we kept the same ratio of categories and products, we split the NewChic dataset into three different sizes. We shrunk it down to

Table 4: NewChic Split Dataset Accuracies

Dataset Size	Naive Bayes	Logistic Regression	Decision Tree	SVM
18750	74.95%	92.85%	72.75%	94.1%
37500	75.85%	92.85%	74.80%	94.05%
75000	90.90%	94.00%	82.95%	94.00%
Averages over n=20 tests, Percent Testing Data Withheld = 20%				

both 18,750 products and 37,500 products, and tested on the full dataset of 75,000 products as well. As shown, the accuracy of these sub-datasets increase as the quantity of products increases. As seen in Figure 5, the accuracy of the full NewChic dataset was averaged from the subcategories as well, not just categories. The subcategories had a huge disparity compared to the regular categories, showing very small average accuracies. For example, with the SVM classification method, going from categories to subcategories brought the accuracy from 90.90% to about 70%. This trend continues for the rest of the classification methods, showing that they decrease dramatically.

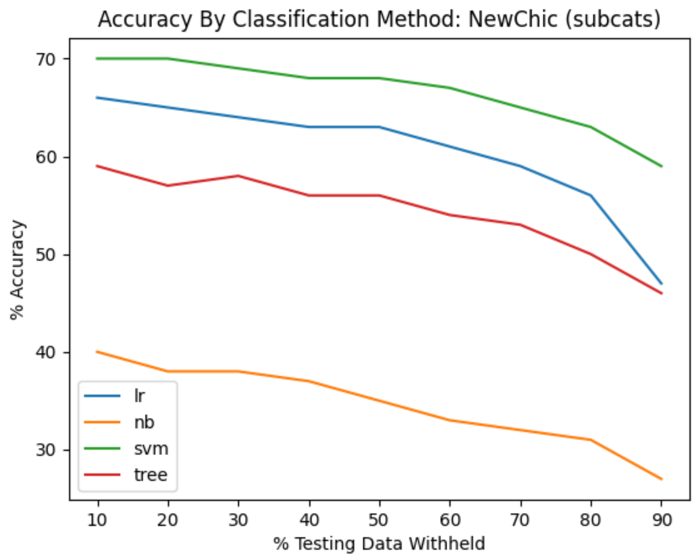


Figure 5: NewChic Datasets (trained on subcategories)

From these tables and graphs, we can infer that smaller datasets tend to have lower accuracies. As the size of the dataset increase, the average accuracy increased as well. We can also infer that as the amount of categories increases, the accuracy will inversely decrease. Due to there being more diversity and features as categories increase, the classifier will have a more difficult time correctly guessing the exact category correctly. Intuitively, larger datasets also provide more data for the classifier to train on; assuming this data is reliable (i.e. it doesn't conflict with previous tf-idf representations or contain miscategorizations), it means the classifier gets more data to 'learn' on, which improves its ability to predict.

4.2.2 Which classification method is analytically better or worse?

Since four classification methods have been used in the project, it might be beneficial to understand which method is better or worse. We particularly analyze which methods yield the best accuracy, and the amount of time it takes for the methods to train on the dataset. To ensure correct and comparable classification results, all methods are trained and tested on the same data. The data was split in 80% training data and 20% testing data.

The graphs in Figure 6 demonstrate the accuracy of each classifier on the datasets. The x-axis represents the percentage of data that was withheld from the training data for testing, and the y-axis represents the reported accuracy based on classifying the training data. The classifiers are producing similar trends for all datasets. All of the tables and graphs indicate that the SVM classifier is generally the most accurate, followed by Linear Regression, then Naive Bayes, and finally the tree classifier method. The decrease in accuracy as the testing data % withheld increases is expected, since the classifier is exposed to less data to learn on, diminishing its ability to predict.

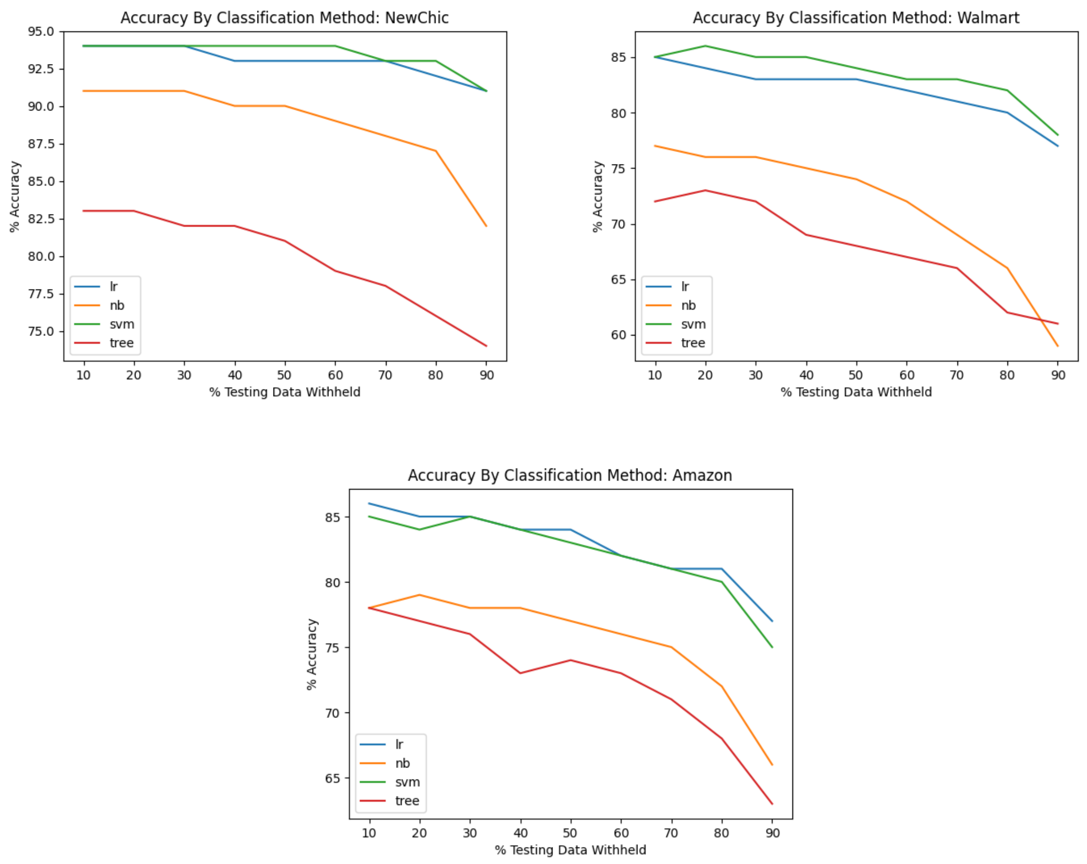
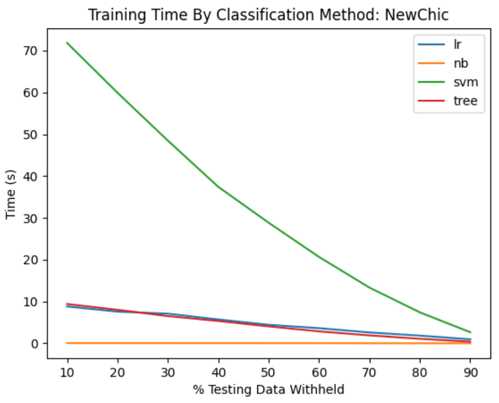
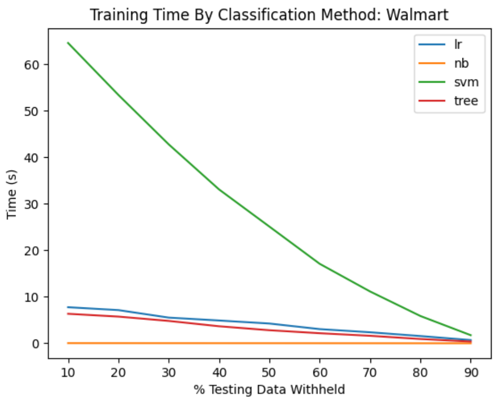


Figure 6: Accuracy vs Testing Data Withheld Plots

The graphs in Figure 7 demonstrate the time it takes for each method to perform classification on the datasets. The x-axis represents the percentage of data that was withheld from the training data for testing, and the y-axis represents the time it takes to classify the testing data. The trends again are quite similar across all three datasets. These results illustrate an advantage of the less-accurate methods seen above in figure 6: Naive Bayes (the second-least accurate method) takes the least time to train by far, followed by Linear Regression, then tree classifier, and finally the SVM method.

The results indicate that the most accurate classifiers (i.e. SVM and Logistic Regression) take the most amount of time to run, whereas the less-accurate classifiers (i.e. decision tree and Naive Bayes) take the least amount of time to run. Taking both





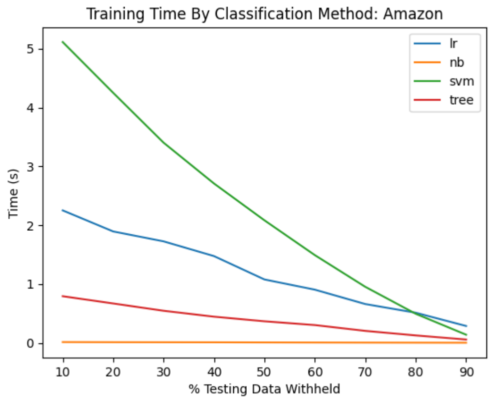


Figure 7: Training Time vs Testing Data Withheld Plots

speed and accuracy into account, it would seem that the Linear Regression method – which was the second-fastest and second-most accurate) constitutes a suitable compromise (though the impressive speed of Naive Bayes makes it a viable option in many cases as well).

4.2.3 How accurately does a classification method classify on datasets other than the trained dataset?

Typically, a classifier is trained based on one dataset, and then tested for accuracy based on values from the same dataset (or random, manually-inputted values from the user). This is fine for testing its theoretical efficacy, but in practice the classifier would be exposed to items and values it might not have encountered. As such, we also decided to test whether training a classifier on one dataset will work with testing data from different datasets as expected; by extension, this question pertains to products that are not represented in the training dataset at all (e.g. if the user inputs 'pan' when there is no cookware category).

We trained a classifier on all four methods (i.e. Naive Bayes, Decision Tree, Linear Regression, and SVM) using either the NewChic or Walmart dataset, and then tested classifying products on the other dataset which had matching categories. For example, since both datasets have the category 'house', we might train the dataset on the 'house' elements in the NewChic dataset, then input all products in the Walmart dataset that are also in a 'house' category and observe how well the classifier performs.

Table 5: Dataset Sizes

Trained Dataset	Tested Dataset	Category Tested	Classifier Method	Average Accuracy (%)
NewChic	Walmart	Beauty	Naive Bayes	40.277
			Logistic Regression	52.163
			Decision Tree	29.906
			SVM	45.525
		House	Naive Bayes	90.285
			Logistic Regression	90.182
			Decision Tree	90.092
			SVM	90.793
Walmart	NewChic	Beauty	Naive Bayes	0.053
			Logistic Regression	16.554
			Decision Tree	14.336
			SVM	14.32
		House	Naive Bayes	1.357
			Logistic Regression	22.553
			Decision Tree	11.398
			SVM	13.673
Averages over n=25 tests, Percent Testing Data Withheld = 30%				

Table 5 shows the datasets tested and trained, the category tested within both datasets, and the classification method and average accuracy. The accuracy is recorded by training a classifier on the trained dataset, and then taking all of the products inside of the testing dataset that equal the category, and seeing how well the classifier can put the products inside of their original category. For example, the first row shows a classifier trained on the NewChic dataset, and then every product inside of the Walmart dataset with the category of 'Beauty' is run through the classifier. The accuracy increases if the classifier successfully predicts the same category, eg. Beauty, and decreases if it predicts incorrectly.

As seen in Table 5, two categories were tested - 'beauty' and 'house' (for 'house', we also included 'Household Essentials' and 'Home' categories). Accuracies trended higher on trials when trained on the NewChic dataset and tested on the Walmart dataset, and lower when the datasets were reversed. The NewChic classifier saw accuracies between 29.906% and 52.163% for the beauty category, and between 90.092% and 90.793% on the house category. On the contrary, for the Walmart classifier, the beauty category found accuracies between 0.053% and 16.554%, and accuracies between 1.357% and 22.553% for the house category.

These results raise a question - why is the NewChic dataset much more accurate than the Walmart dataset? Our theory is that the accuracy may depend on whether the tested dataset can be defined as a subset for the trained dataset. For example, as shown in Table 2 and Figure 3, the NewChic dataset has a much higher size of products than the Walmart dataset. Furthermore, the 'beauty' and 'house' categories on the NewChic dataset are also much higher than the respective categories on the Walmart dataset. Knowing this, we surmise that there are many more products per category to train a classifier on in the NewChic dataset, meaning the smaller amount of products in the Walmart dataset will have more data to base its predictions on. Inversely, the Walmart dataset classifiers will have little data for these categories and much more data from other irrelevant categories, so the classifier will have a much harder time predicting for the NewChic dataset.

Using this information, we can also infer the answer to our question three hypothesis is that using trained classifiers on different sizes or categories of datasets has the possibility to be accurate, however, training one dataset does not signify that the model will act the same on datasets other than itself.

4.2.4 How robust are the classification methods?

In order to measure the robustness of the methods, we set up two experiments with both the NewChic and Amazon datasets.

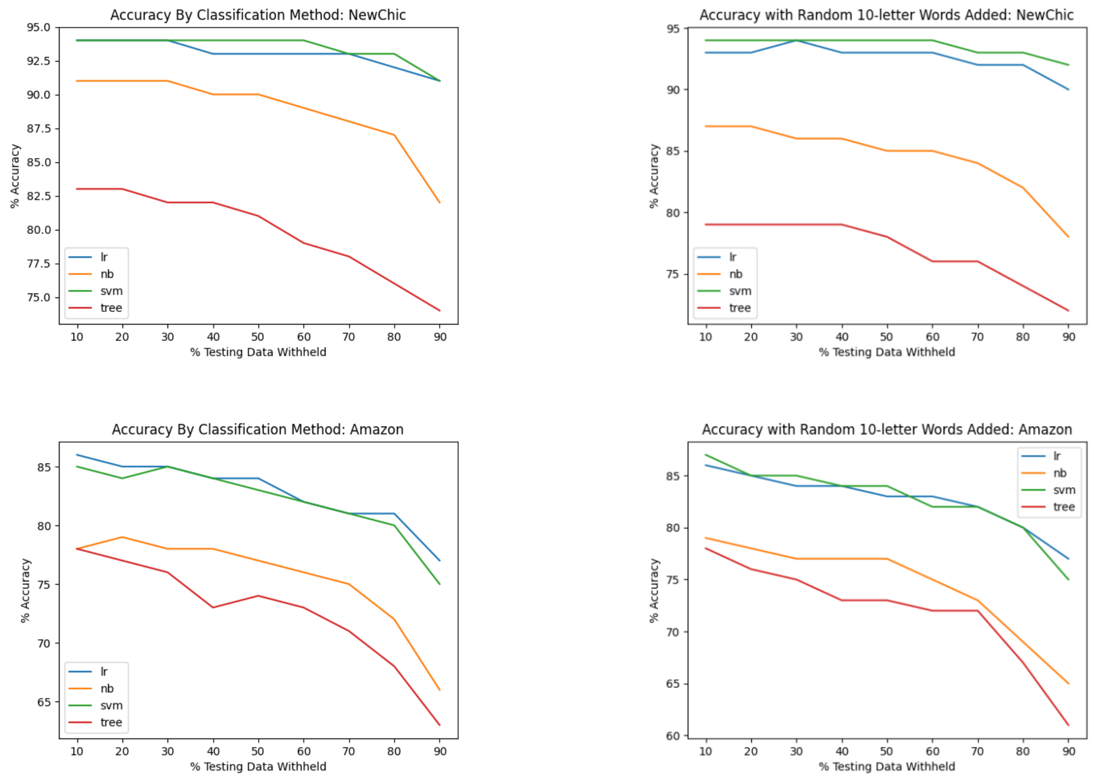


Figure 8: Results After Adding Random Words to Dataset

Our first experiment tested the effect of adding a random 10-letter word to the end of each product name in our dataset. As mentioned above, the random word is comprised of 10 alphabetical letters, and appended to the end of each product's name. Figure 8 shows the results before and after adding the random words to each dataset.

After the words are added, there is a minor increase in the accuracies and the trend changes. Since each word added to a product is random, then there should be no pattern with respect to the new words in each category. Therefore, if the methods are robust to the added noise, much impact is not expected. If, for example, the classifier was putting extra importance on those random words during training, then the accuracy on the test set would suffer since none of them would have the random words. The main idea shown in these figures is to show that the classifiers are robust in the presence of random noise.

Our second experiment tested the effect of removing the 10 most common product words from our dataset. It is important to note that our models utilize a tf-idf representation, which places a weight to each word and signifies the contextual importance of the word in the dataset. For instance, words such as 'the' and 'and' are less important than words related to the category itself. The top 10 most common words were removed before the tf-idf implementation. Table 6 shows the top 10 most common words which were removed from the datasets.

Table 6: Top 10 Most Common Words Removed

	NewChic dataset	Amazon dataset
Words	for - 9765	the - 841
	men - 8816	of - 760
	casual - 7461	set - 712
	with - 6426	pack - 664
	bag - 6116	and - 656
	leather - 6064	with - 574
	printed - 5735	scale - 564
	women - 4881	figure - 541
	shirt - 4387	model - 537
	cotton - 4276	for - 461

Figure 9 shows the effects of removing the top 10 most common words from the NewChic and Amazon dataset. The trends remain similar for both the datasets. The accuracy range also remains similar for the Amazon dataset. However, the accuracy range decreases for the NewChic dataset.

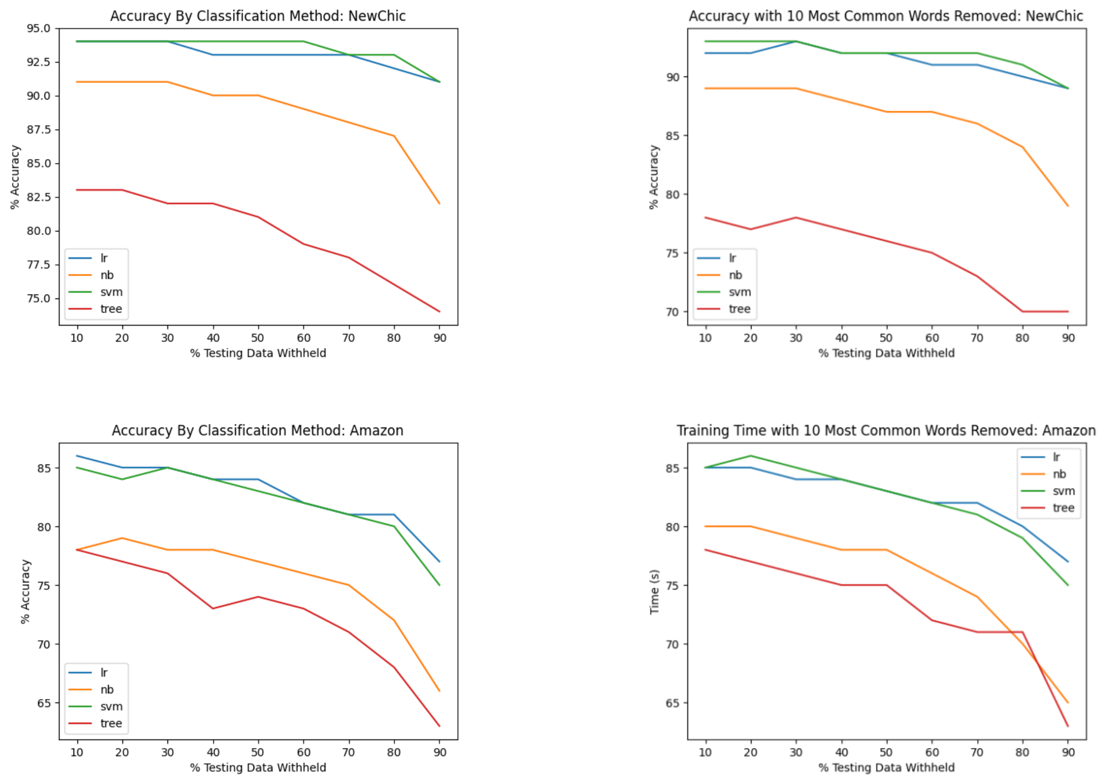


Figure 9: Results After Removing 10 Most Common Words in Datasets

We believe the reason for the decrease in accuracy in one, but not the other is related with the types of words used. The list of omitted words for the Amazon dataset consist of words such as 'of', 'the', 'and' and 'with', which aren't likely to help the classifier determine what category the product should fit in. Hence, the accuracy remains the same. However, the types of words being omitted in the NewChic dataset are words that are closely related to some categories in the dataset, such as 'bags', 'men', and 'women'. These kind of words are much more important for classification, so it makes sense that their omission causes a decrease in accuracy.

5 Ethics and Legal Practices

Listed below are the ethical and legal considerations addressed in our project:

1. The three datasets used in our analyses are publicly available, and were obtained on data.world.
2. The information used in the datasets does not contain any type of information which can can infer the true identity of anyone has bought the products.
3. During the use of our UI application, we will not collect any information related to personal privacy, and will not record the information entered by the user.
4. The research questions that were examined in this work are novel and have not been adopted from previous research work.
5. The code fragments such as the classification methods used from the SciKitLearn libary are properly documented in the code.

6 Effort Sharing

Our joint and individual contributions are shown below in Table 7.

Table 7: Effort Sharing

Joint Efforts	Pranjal Atrey	Alex Kisiel	Jordan Le	Erik O'Hara
20%	20%	20%	20%	20%

Pranjal Atrey's Contributions. Pranjal directly worked with the datasets, from downloading them to performing preliminary analysis and retrieving information (i.e. max/min/average values, # of unique products, etc). He also implemented the tree decision classifier and logistic regression methods into the code. In addition, Pranjal implemented evaluation techniques such as confusion matrix and class weight implementation to ensure that the datasets were balanced, and that classification was performed correctly. He also altered the datasets which were used in our study for checking robustness (RQ4).

Alex Kisiel's Contributions. Alex implemented the Scikitlearn libraries which vectorized and embedded datasets into tfidf numerical representations. In addition, he helped with the dataset stratification process, as well as formatting the data to be compliant with classi.py. Alex also worked on testing, including implementing the 'classifier pickling' process and writing the main testing functionality within train(). He also assisted by developing the graphs that display accuracies and training times for different classification methods, which can be seen throughout Section 4. Alex also implemented the Naive Bayes classifier method, which is found in the train() method of classi.py.

Jordan Le's Contributions. Jordan implemented the support vector machine (SVM) classifier method as well as analyzed different parameters, including different possible kernels. In addition, Jordan worked with the data, specifically on refactoring the datasets for multiple purposes including fixing imbalance issues, splitting the datasets for RQ1 analysis while ensuring the ratios are equal, and removing unnecessary feature columns which weren't used in our study (e.g. unique_ID).

Erik O'Hara's Contributions. Erik translated the NewChic dataset (originally in French) and created a user interface for the end-user to assist in training and classification of datasets. Erik also created and ran scripts to train and classify multiple datasets at once, while also receiving the average accuracies, variances, and standard deviations of the values. These also include the scripts required for research included in RQ1 and RQ3, specifically to retrieve only certain categorical products from each dataset and testing the accuracy of another, along with the analysis of both questions and their outcomes.

Joint Contributions. As a joint effort, all contributors worked on analysis of the four research questions and discussed the conclusions together. In addition, all contributors worked together to write the classi.py file, which contains the functions to train and classify the models.

7 Conclusion and Future Work

Based on the research questions and results obtained studied in this work, we conclude the following:

1. RQ1: How does the size of a dataset affect the accuracy of the classifier?
   (a) Smaller datasets tend to have lower accuracies.
   (b) As the number of categories increase, it can become more difficult to differentiate between them, resulting in lower accuracy (particularly if the categories are closely-related).
   (c) Many factors contribute to accuracy, such as features and data-element/category diversity.
2. RQ2: Which classification method is analytically better or worse?
   (a) SVM and Logistic Regression have the best accuracy but take longer.
   (b) Naive Bayes and Decision Tree take the least time but depict worse accuracies than other methods.
3. RQ3: How accurately can a method classify on datasets other than the trained dataset?
   (a) Training one dataset does not signify that the created model will generalize on other, separate testing datasets.
   (b) Accuracy of these methods may depend on whether the tested dataset can be a subset of the trained dataset.
4. RQ4: How robust are the classification methods?
   (a) The classifiers are robust in the presence of random noise, and can depend on the type of noise.
   (b) Removing contextually-important words in a dataset affects classifier accuracies more than removing less-important words.

There are many future research avenues for projects similar to ours. One such opportunity would be to allow users to compare their classifications to those of other users; in doing so, users would be able to list their classifier results alongside those of other users to compare accuracies and explore other, potentially more suitable categories. Another potential opportunity for further analysis is looking at the top 3 highest confidence category matches returned by each classifier in order to better understand the dynamics and inner function of the classification models. Finally, of course, there is always room for improvement with regards to optimization: using parameters in the ScikitLearn functions and other algorithmic improvements can increase prediction accuracy or reduce training time, and make classifiers even more useful.

References

[1] Amazon. Create inventory file templates and classify your products. https://sellercentral.amazon.com/gp/help/external/200956770, accessed 09/2020.

[2] Marin De Beauchamp. Product categorization: classical machine learning problem for a difficult e-commerce task. https://medium.com/manomano-tech/product-categorization-classical-machine-learning-problem-for-a-difficult-e-commerce-task-b5a9039d884b, accessed 09/2020.

[3] Maggie Yundi Li, Stanley Kok, and Liling Tan. Don't classify, translate: Multi-level e-commerce product categorization via machine translation. https://arxiv.org/pdf/1812.05774.pdf, accessed 09/2020.

[4] Avinash Navlani. Support vector machines with scikit-learn. https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python, accessed 09/2020.

[5] Sushant Shankar and Irving Lin. Applying machine learning to product categorization. *Department of Computer Science, Stanford University*, accessed 09/2020.