Food Item Category Classification Using Probabilistic Models

Morgan Nañez
mn9794@princeton.edu

Abstract

The purpose of this project is to apply probabilistic models to achieve a classification task. Given the title of a food product, classify which grocery category it belongs to. The input data will be a text name, that is then preprocessed into a vector. The model takes this vector and returns which class it is most likely to be in. Each of the models are supervised and have been trained to classify 95 different grocery categories, from 'chocolate' to 'asparagus'. I attempt this classification task using four different probabilistic models. The Gaussian naive Bayes and logistic regression are probabilistic models that attempt to derive the posterior probability $P(class|features)$. I also design two neural networks that out preform both logistic regression and Naive Bayes. One of the biggest hurdles I face in the modeling process is the intrinsic class imbalance within the input data.

1 Motivation

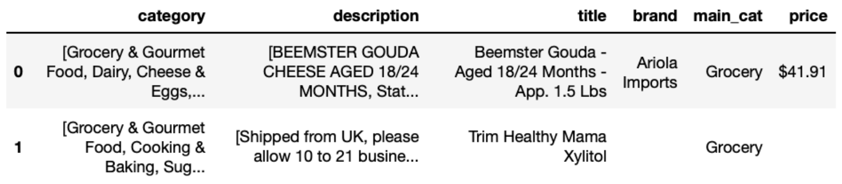
Over the course of the semester, we have been presented with various probabilistic modeling techniques ranging from HMM's to VAE's to neural networks. I was most curious about how I can use data in a supervised setting to extract posterior probabilities of different classes. While regression in itself is useful, I wanted to tackle probabilistic modeling as a tool for classification. The motivation for this project is a part of much larger project where I plan to produce the total carbon gas emissions of a grocery shopping cart. In order to do so, I first must be able correctly categorize each food product, to obtain its associated green house gas emission. The following project explores different models that take a food product name as an input, and classify it into its correct grocery food group. My hope was to not only produce a functional model but also discern the strengths and weaknesses between each of the models tested.

2 Data

For this project, I am using public Amazon Metadata. For the purpose of my project, I limit my input data to just be from the 'Grocery' genre of products. The data set is quite expansive, roughly 300,000 data points and has details about price and technical details for showing the products online, such as html descriptors. For the purpose of this project, I kept the following columns: `'category'`, `'description'`, `'title'`, `'brand'`, `'main cat'`, `'price'`

It is important to note that because this data is coming from Amazon.com, and not associated markets, such as Whole Foods, so many of the products in this database are shelf-stable or gift type grocery items, such as chocolates or spices. There is lack of fresh produce or dairy items represented, which will be discussed later in the paper.

Figure 1: Raw Input



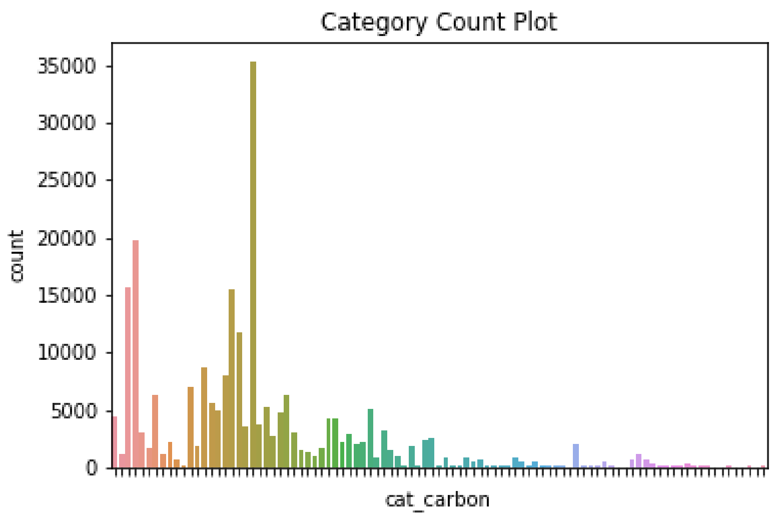
	category	description	title	brand	main_cat	price
0	[Grocery & Gourmet Food, Dairy, Cheese & Eggs,...	[BEEMSTER GOUDA CHEESE AGED 18/24 MONTHS, Stat...	Beemster Gouda - Aged 18/24 Months - App. 1.5 Lbs	Ariola Imports	Grocery	$41.91
1	[Grocery & Gourmet Food, Cooking & Baking, Sug...	[Shipped from UK, please allow 10 to 21 busine...	Trim Healthy Mama Xylitol		Grocery	

2.1 Data Exploration

Because I am using a supervised learning approach to train the model, I want to see what unique categories I have in my data. Over the entire data set, there are 1353 unique categories being mapped to. The categories provided in the data set can be very specific, resulting in many categories with fewer than 10 data points.

It is important to note that most categories have lower value counts, with fewer than 500 corresponding points. From further exploration I found that many of these lower count categories actually belonged to other categories. For example 'Beer' and 'Alcohol' are part of the same grocery food category but 'Beer' only has a single data point. For supervised learning, it's important that each category is represented adequately in the data set. It would be hard to accurately learn features for a category that only has one example data point. To remedy this issue, I merged lower count categories into larger categories. This creates fewer overall categories to map to as well ensuring that each category is adequately represented in the training processes. This resulted in a mapping of all 1353 categories to 95 new categories.

Figure 2: Distribution of Data

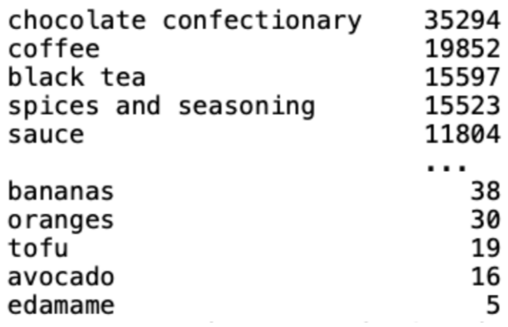


In the plots above is it very clear to see that there a large class imbalance. The largest classes have tens of thousands of data points, while the smallest classes have tens. This class imbalance is a result of the origin of the data, as explained above. We see that the smallest classes are fruits and vegetables, which are not typically bought on Amazon.com, whereas chocolate is a common commodity often bought for occasions and gifts.

2.2 Data Cleaning

Because I am working with raw text meta data, it is imperative to clean and standardize the input data. I gather a list of stop and filler words, such as 'the' and 'because' that will be removed from each

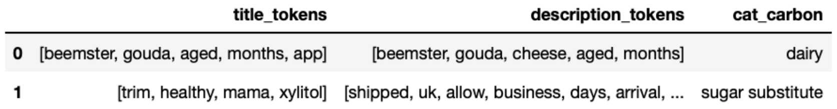
Figure 3: Category Counts



```
chocolate confectionary    35294
coffee                     19852
black tea                  15597
spices and seasoning       15523
sauce                      11804
                           ...
bananas                       38
oranges                       30
tofu                          19
avocado                       16
edamame                        5
```

input text. I also remove all numbers. I also transform each text into lower case as well as remove any FDA standardized text that is commonly found on food products. Additionally, I drop the ’brand’ and ’price’ column as my focus started to hone in on ’title’ text. The main reason for this is that if I were to train a model then test it on a different data set, outside of the Amazon Metadata database, I may not have access to brand name or price, but I know I will have access to the title of the product being purchase. This allows the model to be used in variety of external settings. Compare the raw to the cleaned description text below.

Figure 4: Cleaned Input



	title_tokens	description_tokens	cat_carbon
0	[beemster, gouda, aged, months, app]	[beemster, gouda, cheese, aged, months]	dairy
1	[trim, healthy, mama, xylitol]	[shipped, uk, allow, business, days, arrival, ...	sugar substitute

You can see that the cleaned data has only the most important aspects of the original input. In removing some words, I should note that I have also removed some potential context. This limits potential feature engineering from being fully optimized, such at BERT or other transformers that are aided by context around a word. I made the choice to ignore context around the word and spatial location of the word as I believe since the title is short in length, everything in it, after processing, is equally important. I am aware that overall, this may hurt the model, as now ’zucchini bread’ and ’zucchini with bread’ now have the same embedding, even if their categories are different. I chose to take this risk and acknowledge that this is an area for improvement for later iterations of the project.

3 Feature Engineering

For this approach, I downloaded google’s pre-trained ’genism’ Word2Vec Model that outputs a vector for each word. I choose this model because it directly allows one to find similarity between words. For example, Word2Vec model is able to say that the most similar words to ’sweet’, in our data set, are ’tangy’, ’fruit’, and ’sweetness’. The motivating idea is that words belong to the same category with have similar Word2Vec vectors. When creating the Word2Vec model I had two choices, to train the model myself on the training set title tokens, or use a predefined model and risk not having all the words in my training corpus. While training my own model has its benefits, such as each of the word vectors are task specific to classifying grocery products, after running preliminary models with both Word2Vec variations, I found that using a pre-trained model worked better. I believe that is due to the fact that the pre-trained model is more expansive and expressive and can capture a wider range of words, that may be present in the test set and not the training set.

To create a final vector for each sequence of title tokens, I take the average of all Word2Vec vectors present in the input. I then use Scikit-learns ’StandardScalar’ method to standardize features by removing the mean and scaling to unit variance. This method eliminates all contextual and spatial dependencies within the title input. The output of the Word2Vec model is 300 features, which means that the number of features my model is working with is 300.

4 Probabilistic Models

For each of the models I present, I utilized supervised learning. I separate my data into 20% testing data and 80% training data. By using probabilistic models, I hope to define a posterior probability between the input title tokens, which have been transformed into features, and each of the potential classes, which are food categories.

4.1 Gaussian Naive Bayes

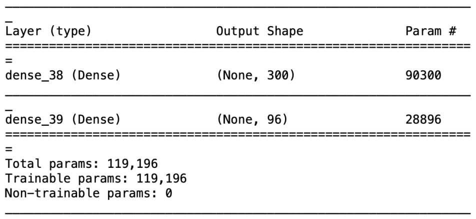
Gaussian Naive Bayes is a probabilistic model that applies Bayes theorem, in conjunction with the idea that features within classes are independent. The model assumes that each feature is an independent contributor to the class probability. We use this model to calculate the posterior probability $P(class|features)$ given $P(class)$, $P(features)$, and $P(features|class)$. I used a Scikit-learn's Gaussian naive Bayes model, which assumes the underlying likelihood of the features to be Gaussian. While I do not believe this model will preform the best, I will use it as a baseline to compare other models to.

4.2 Logistic Regression

I utilized Scikit-learns Logistic regression model, using L1 Loss and the lib-linear solver. Because there are 300 features, L1 regularization will reduce extra noise by giving less weight to unimportant features, eventually causing some weights to go to zero. When the weights go to zero, this effectively makes the feature non-beneficial too the task being learned. While I am tackling a classification problem, logistic regression is, as its name implies, a regression model. Logistic regression applies a one-versus-rest approach to each class, and a sigmoid function, to produce a probability over each of the classes. One of the useful facts of logistic regression is that it does not assume a linear relationship between the features and classes. This assumption allows the model to create more intricate relationships, ideally making the posterior estimation stronger. The model I use uses maximum likelihood estimation to estimate model parameters.

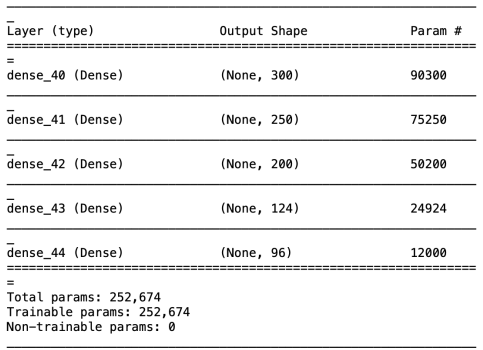
4.3 Neural Networks

When testing neural networks, I tried a very simple, one hidden layer network, and a more complex model, with 4 hidden layers. Both Neural networks use adam optimizer, sigmoid activation, and cross entropy loss. Similarly to logistic regression, neural networks can learn nonlinear relationships. While the two models are similar, neural networks can be more complex and allow for deeper relationships to be learned. However, it is harder to trained a neural network, as they are prone to overfitting. When designing the networks, I tried a variety of different optimizers, such as 'adagrad' and 'rmsprop', but found 'adam' to be most successful. For the execution of the code, I used keras to train and fit the models to the training data. Both models were trained for 5 epochs. Below are the model architectures I used.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_38 (Dense)             (None, 300)               90300
_________________________________________________________________
dense_39 (Dense)             (None, 96)                28896
=================================================================
Total params: 119,196
Trainable params: 119,196
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Simple Neural Network $NN1$



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_40 (Dense)             (None, 300)               90300
_________________________________________________________________
dense_41 (Dense)             (None, 250)               75250
_________________________________________________________________
dense_42 (Dense)             (None, 200)               50200
_________________________________________________________________
dense_43 (Dense)             (None, 124)               24924
_________________________________________________________________
dense_44 (Dense)             (None, 96)                12000
=================================================================
Total params: 252,674
Trainable params: 252,674
Non-trainable params: 0
_________________________________________________________________
```

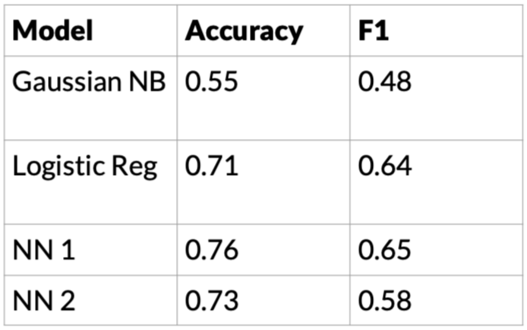
Figure 6: Four Layer Neural Network $NN2$

I wanted to compare both neural networks because I believed that even though the larger network has more parameters and is more complex, it will be more susceptible to overfitting, resulting in a worse test accuracy. I restrain from using other neural networks, such as convolutional or recurrent neural networks, because both of these networks imply some spatial dependency within the input data. While this dependency exists in some NLP and classification tasks, I do not think this dependency is as important in my post-processed data. Each input, in size, is rather short and there is not much additional context.

5 Results

In the figure below, 'NN_1' refers to the single hidden layer model and 'NN_2' refers to the the four hidden layer network. I report both the accuracy and F1 scores that the models produced on the testing data set, respectively.

Figure 7: Results



Model	**Accuracy**	**F1**
Gaussian NB	0.55	0.48
Logistic Reg	0.71	0.64
NN 1	0.76	0.65
NN 2	0.73	0.58

From the chart, we see that the simple neural network performed the best, with both the highest accuracy and the highest F1 score.

This confirms my intuition that the more complex neural network did worse than the simple model. Logistic regression preforms within range to the neural networks, as expected to some degree since it can handle nonlinear dependencies. Gaussian Naive Bayes performs the worst in both accuracy and F1, which leads me to believe that the features are not independent of one another. This is not a surprising result due to the nature of the input data. Word2Vec is a vector of 300 feature representing a word as whole. It is feasible to believe that there are at least two features that together lead to the determination of a class. For example if the vector was apple = [1,1,...], where the first index is represents 'round' and the second index represent 'red', that you would need these feature together to discern from an orange or salmon.

It's also important to note that due to the class imbalance, even if the model only predicted 'chocolate', it would still receive an accuracy of 10% which is significant. The class imbalance can lend false hope that our model is performing well, when in reality it has only learned five of the 95 categories. It is important to explore how the models does on all classes rather the most dominate ones.

5.1 Class Imbalance

Because there was such a striking class imbalance in the the training and testing data, I wanted to explore how the best and worst model did on the some of the most plentiful categories and some of the categories with fewer data points. Below, I compare accuracy and false positive rates within different categories.

For classes that have the most data points, both models did well in terms of accuracy, and have lower false positive rates. This leads be to believe that there was enough data to learn relationships between features and these classes. For classes with few data points, the results between accuracy and false positives have high variance and I can't make any broad meaningful conclusions. Since the accuracy for 'tofu' in the simple neural network is 1, perhaps that features representing tofu are very unique and have a greater dissimilarity between other classes. However, for a classes like 'banana', where there are other classes of fruits, say 'apple' or 'oranges', it may confuse the model, leading it to not have any predictions.

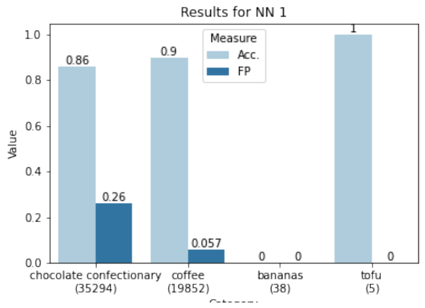


Figure 8: Simple Neural Net

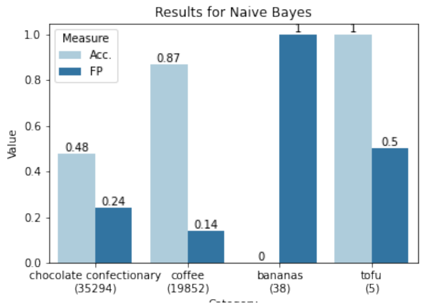


Figure 9: Gaussian Naive Bayes

6 Conclusion

In conclusion, the simple neural network worked best, out preforming all other models in both accuracy and F1 metrics. On the other-side, Gaussian Naive Bayes preformed the worst across all metrics.

Some short comings with this project is that even at its best, my highest accuracy was less than 0.80. While this could be attributed to poor model selection, I think the more likely culprit is the pre-processing and featurization of the data itself. Exploring different word embedding techniques, such as BERT, and feature selection tools could make the input data itself stronger, therefore making the models more useful and increasing performance. I would also like to explore adding in price and brand name into the feature space. It's logical to assume that certain goods are always priced higher or lower compared to other goods, such as avocados or caviar. However, this could cause issues when extrapolating the model to other data sets. Weight of an item may also play a role in price, which leaves me hesitant to make a price part of the feature space. Another glaring setback is the class imbalance intrinsic to the data. I could try to separate these classes into smaller categories, however this would be much more intensive compared to the merging process. Another way to combat this issue is generating more data for classes with fewer data points using different generative or sampling techniques. Overall, I see that there is room for improvement. I am keen on exploring how the role, if any, of spatial dependencies can be used in either prepossessing and feature extraction, or within the models themselves, such as RNN's or CNN's. I would also like to expand on the current neural network models, such as adding dropout and experimenting with other varieties of activation functions.