In Class Activity 4

Text classification with Naïve Bayesian Model

In today's activity we are going to classify some text using a naïve model. You'll observe that the model is called "naïve" because it is looking at word counts and just word counts. The model does not consider the proximity of one word to another, the context of the words, or the sequence in which the words are used.

To make our process work, our text will need to be broken down into a collection of units (called ***tokens***) and those tokens are arranged in a pattern called a ***vector***. This is a crude example of what happens when LLM service (like ChatGPT) receives a prompt from a user such as yourself.

Our imaginary scenario is that of an automated service request system. Requests for service and assistance come in as "trouble tickets". Our script will build a Naïve Bayesian model and then use that model to route the request to the correct "expert". You can imagine that there is one expert for each department.

Here are the departments in our imaginary organization.

1. accounting
2. customer support
3. human resources
4. information technology
5. marketing
6. operations
7. sales

A note to students! In the text that follows, we will look at the Python libraries we will use in today's script.

Your dear old instructor is not concerned with your ability to memorize and recall the exact names of Python libraries.

However, looking at the libraries used *is* a nice way to understand what we are trying to do in the script at a high level.

Our script will use these libraries:

1. **pandas** – this library for handling data in DataFrame format. You can think of a data frame as something like a database table, or an Excel sheet. It has rows and columns. But, the DataFrame data format has lots of handy features built into it to make writing code easier.

2. **train_test_split** – this library is for splitting data into training and testing sets. Supervised Machine Learning models usually depend on breaking the data into two sets. The first set is used to build the model … this is the "training" set. The other set is used to validate the model.

Validation of the model is important as it is the only way that we can approximate how good the model will be in the future when it is working with new data.

3. **TfidfVectorizer** – this library is for converting text data into TF-IDF feature vectors.

   What's that you ask? It's just a fancy way to look at the "Term Frequency' (the number of times a word appears in a document) relative to how often the word appears in all the documents together. The combination of all the documents together is called "the corpus".

   - **Term Frequency (TF):** Term frequency measures how often a term (word) appears in a document. It's calculated as the number of times a term appears in a document divided by the total number of terms in the document. This is a measure of frequence within a single document.
   - **Inverse Document Frequency (IDF):** Inverse document frequency measures the importance of a term in the entire corpus. It's calculated as the logarithm of the total number of documents in the corpus divided by the number of documents containing the term. Why use logarithms? Because statisticians are a wacky bunch, that's why!

     Seriously… when the decimal values less than 1 but greater than 0 get really, really close to zero, it is often handy to transform them in this way. It makes comparison of these tiny values easier.
   - **TF-IDF Calculation:** TF-IDF is computed by multiplying the term frequency (TF) of a term by its inverse document frequency (IDF). This results in a higher TF-IDF score for terms that are frequent in a document but rare in the entire corpus, indicating their importance in distinguishing the document.
   - **Vector –** We just put all the terms together in a sequence, organized from most frequent to least frequent…. And we call that a vector.
   - **I am personally glad that I did not have to figure all that out for myself.** You should be too!

4. **MultinomialNB** – this library is for implementing the Naive Bayes classifier. The prior step got all my data ready, basically transforming each document into a vector with numeric values. This step builds the model necessary to look at each vector and categorize it correctly. The software first builds the model (with the training data set) and then attempts to determine its accuracy (with the validation data set).

5. **classification_report** – This is for evaluating the classifier's performance. We want to know how our model did with the training set. Here's the output we see:

   - **Precision** is the ratio of correctly predicted positive observations to the total predicted positive observations. In other words, it measures the accuracy of positive predictions. It is calculated as:

     $$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives}+\text{False Positives}}$$

- **Recall** is the ratio of correctly predicted positive observations to all the observations in the actual class. In other words, it measures the completeness or sensitivity of the classifier. It is calculated as:
  $$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- The **F1-score** is the harmonic mean of precision and recall. It provides a single metric that balances both precision and recall. It is calculated as:
  $$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$
  The F1-score combines precision and recall into a single metric that balances both metrics. It provides a way to assess the overall performance of a classifier, considering both false positives and false negatives.

  If the F1-score is close to 1, it indicates high precision and high recall, meaning the classifier is performing well in terms of both minimizing false positives and false negatives.

  If the F1-score is close to 0, it indicates poor performance in terms of either precision or recall, or both.

  If the F1-score is closer to 0.5, it suggests that the classifier is achieving a balance between precision and recall. It is ok, but there is still room for improvement!

  We multiply by two in this equation because we want to end up with a scale that ranges between 0 and 1, not zero and 0.50.

- **Support** is the number of actual occurrences of the class in the specified dataset. It represents the number of samples in each class. Support is just a count of occurrences.

These metrics provide insights into how well a classifier is performing for each individual class.

6. **matplotlib.pyplot** – This library is just for visualization. We'll use it to make some bar charts that suggest what our vectors look like for each department.

Instructions

1. Students should download the activity4.zip file and unzip it into their mis3536workspace. (There are two files, the Jupyter notebook, and the data file.)
2. Explore the data source (it's a CSV file this time!) Be careful not to edit it.
   a. Based on what you see here, what ***data cleaning*** has been done?
   b. Based on what you see here, does this look like we are headed towards a supervised model or an unsupervised model? Why?

3. Today, we are going to run the Jupyter notebook one cell at a time.
    a. The first cell -- This code essentially loads data, prepares it, splits it into training and testing sets, converts text data into TF-IDF features, trains a Naive Bayes classifier, makes predictions, and evaluates the classifier's performance.
    b. The second cell -- This block of code generates a series of horizontal bar plots, each showing the top 10 most important features for a specific department/class, along with their corresponding coefficients. These visualizations help in understanding which words are most influential in predicting each department/class in the classifier.
    c. The third cell -- This code allows the user to input a problem description, predicts the top two departments likely to handle the problem, and prints the predictions along with their probabilities.
4. Using the third cell in the script, experiment with the model we just built.
5. DISCUSS: There’s *a lot* going on here. What are your chief takeaways?
6. Using the bar plots as a guide, craft a *good* example of your own. Screen shot it, and upload it to the corresponding Canvas assignment.
7. DISCUSS: Consider these two questions:

    Question 1: What are the flaws in this classification mechanism?

    Question 2: Based on what we know about our classification model, what could we do to improve it?