CSC 2730 Project Report

December 6, 2020

William Daniels, Kelvin Nguyen

**Problem and Motivation**

Companies all over the world have customer support centers to help customers with inquiries. But some companies don’t have the manpower to assign customer service representatives to each customer that has problems.

Therefore, we have considered that such a company, in this case, a banking company, help alleviate strain by providing customer support through a chatbot.

For this purpose, we build models that associate customer inquires to the intent of the customer’s inquiry, or in other words, a classification model that can associate raw text to a certain meaning.

**Data Used**

For training purposes, we use a dataset made specifically for classification of banking inquiries and modify it for our problem, only retaining the target intents necessary. We have also added more target intents and training phrases to fulfill our vision of a customer service chatbot.

Dataset source: <https://arxiv.org/abs/2003.04807> | <https://github.com/PolyAI-LDN/task-specific-datasets/blob/master/banking_data/train.csv>

**Features/Attributes and Target**

Our data is separated into phrases and intents. Phrases are the training set for our model and are equivalent to the input our customers will be giving the bot. These include questions, inquiries, and even some declarative statements. The intents are the targets of our model, and are the intention classification of the phrase they are associated with. So, for instance, a phrase stating that a customer wants to change account details will be labeled with the intent “change\_acc\_details”, or something similar.

Since raw text in the form of phrases is the only training data supplied to the model, there are no specific features selected as training features, although, it can probably be said that each word is a feature itself.

**Models and Hyperparameters**

Before we put the data into our model, we separate it into an 80-20 split with the *train\_test\_split* function from sklearn, where 80 percent is assigned to training data and 20 percent is assigned to testing data.

Phrase-intent pairings are shuffled before splitting, which ensures that the training data is representative of kinds of data we would see in the test data. (For example, if the phrase-intent pairings weren’t shuffled, the test data would be filled with phrase intents that weren’t present in the training data, and the model would have terrible accuracy.)

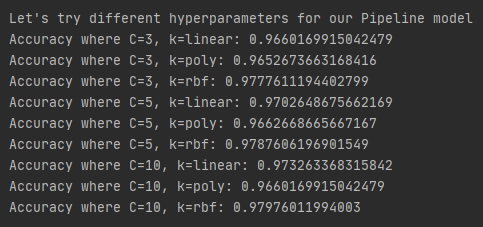
**C-Support Vector Classification w/ TF-IDF Vectorizer**

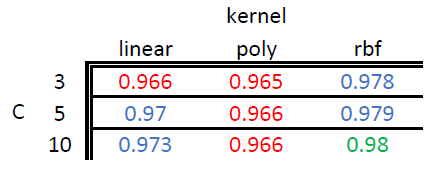
Our first model is a C-Support Vector Classifier paired with the vectorized output of our training.

We create a *TfidfVectorizer* with default hyperparameters and a *SVC* modelwith a linear kernel and supply them to our pipeline model. This model’s accuracy, as displayed through the *accuracy\_score* function (which is the performance measure all models in this report are scored by), is fairly high at around 95 percent.

Let’s try some different hyperparameters.

First, we try the same pipeline model where the ‘C’ parameter in our SVC model is [3, 5, 10, 20] and the ‘kernel’ parameter is [‘linear’,’poly’,’rbf’]. C is the regularization present in our model, which decreases the variance in our model, while also keeping bias at a minimum and ‘kernel’ is (roughly) that which defines how similar two points in a feature space are to each other.

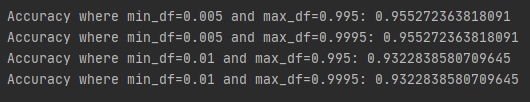


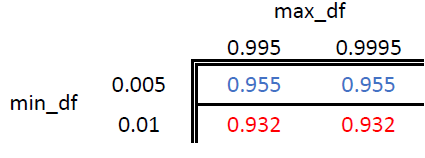


From the graph above we see that the best parameter permutation for our pipeline is a regularization parameter of *C=10* and a kernel parameter of *kernel=rbf.* Why this is the case is explored in the results section of this report.

With the best regularization and kernel parameters chosen, we can alter the parameters of the vectorizer of our data to see if they can supply us with even higher model accuracy.

We change the *min\_df* parameter to [.005, .01], which filters words that have low presence in the dataset, and *max\_df* to [.995, .9995], which filters words that have a high presence in the dataset.

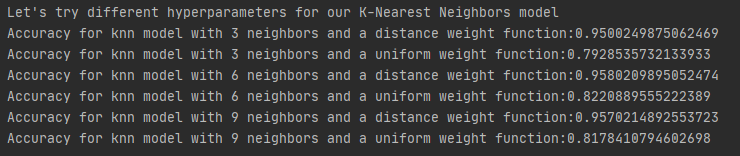




This attempt doesn’t seem to help our model at all, this is explained in the results and discussion section of the report.

**K-Nearest Neighbors w/ TF-IDF Vectorizer**

Our second model is a K-Nearest Neighbors model where we change the number of *neighbors* in the model by [3, 6, 9] and the *weight function* between [‘distance’,’uniform’]. We use the same vectorizer for our data that we use for the pipeline model, without the changes to parameters *min\_df* and *max\_df*.





It seems there is a wide range of accuracy in this model depending on the parameters chosen. Why this happens to be the case is explained in the next section of this report.

**Results and Discussion**

**SVC Model**

Where the regularization parameter C is increased, the model’s accuracy goes up. This is probably due to the shrunken model variance, which avoids overfitting, which, it seems to us, is why the accuracy of the models are higher with this selection. Since the gains made by increasing this parameter diminish over time, making C bigger than 10 will not give such a benefit to accuracy that it would surpass the next best parameter combination.

From worst to best, the kernel parameters of ‘poly’, ‘linear’, and ‘rbf’, give slightly different accuracies. The linear kernel outperforming the poly kernel seems to suggest that the dataset is linearly separable to such an extent that using a polynomial separator does a worse job. And since the radical bias function consistently outperforms these two kernels, it should be the case that this data is not totally linearly separable.

So with the combination of a regularization parameter of *10* and a kernel of *rbf*, we achieve a pretty high classification accuracy of 98%, which is the best of the two models we test.

**TF-IDF Vectorizer Experiments**

It seems that elimination of noise from our dataset is a nonissue, as when the most common and least common of words are removed from the dataset before training, the model accuracy declines by a decent amount.

**K-Nearest Neighbor Model**

The type of weight function chosen for this model clearly have the most impact. While the uniform weight function weighs all points equally, the distance function taxes those points which are farther away from each other, increasing the disparity between them.

For the purposes of this project, it seems weighing points with the distance function doesn’t result in worse accuracy, but the opposite, suggesting that the points the model weighs are placed close together without the distance function implemented, which makes it harder for the model to select the ‘correct’ neighbor.

As for neighbors, a *n* that is closest to 6 seems work the best, although it could be the case that 7 or 8 give a better accuracy. From this we can surmise that a selection of around 6 (or slightly more) neighbors is a great selection for the model to choose from, and on average a *neighbors* parameter of around 6 neighbors contains the best option most of the time compared to other choices for *n*.