

Report for Alphabet Soup Charity

Overview

The non-profit foundation Alphabet Soup wants a tool that can help it select the applicants for funding with the best chance of success in their ventures.

Machine learning and neural networks was used on a CSV file containing more than 34,000 organizations that have received funding from Alphabet Soup over the years to predict whether applicants will be successful if funded by Alphabet Soup.

The data set contains 12 columns of metadata about each organization, which are as follows:

- **EIN** and **NAME**—Identification columns
- **APPLICATION_TYPE**—Alphabet Soup application type
- **AFFILIATION**—Affiliated sector of industry
- **CLASSIFICATION**—Government organization classification
- **USE_CASE**—Use case for funding
- **ORGANIZATION**—Organization type
- **STATUS**—Active status
- **INCOME_AMT**—Income classification
- **SPECIAL_CONSIDERATIONS**—Special considerations for application
- **ASK_AMT**—Funding amount requested
- **IS_SUCCESSFUL**—Was the money used effectively

Evaluate the Models

Initial Model

The target variable for this model is **IS_SUCCESSFUL** which has values 1 for True and 0 for False.

Pre-process dataset

EIN and **NAME** were removed from the DataFrame at the start of the pre-processing process.

The feature variables are as follows:

- **APPLICATION_TYPE**
- **AFFILIATION**
- **CLASSIFICATION**
- **USE_TYPE**
- **ORGANISATION**
- **STATUS**
- **INCOME_AMT**
- **SPECIAL_CONSIDERATIONS**
- **ASK_AMT**

APPLICATION_TYPE and **CLASSIFICATION** was split into subsections through the process of binning and added back to the DataFrame.

When using `pd.get_dummies` it converts categorical data to numeric.

The variables that are not used for the input because they are neither targets nor feature are as follows:

- **EIN**
- **NAME**

Two hidden layers were used. The first hidden layer used 80 neurons and the second hidden layer used 30 neurons. Two types of activation functions were implemented.

Activation function “relu” (activation=**relu**) was used in the first and second hidden layer and activation function “sigmoid” (activation=**sigmoid**) was used in the output layer.

```
[14] # Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
# # Define the model - deep neural net
number_input_features = len(X_train[0])
hidden_nodes_layer1 = 80
hidden_nodes_layer2 = 30

nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(
    tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation="relu")
)

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation="relu"))

# Output layer
nn.add(tf.keras.layers.Dense(units=1, activation="sigmoid"))

# Check the structure of the model
nn.summary()
```

Reasons for using Sigmoid and ReLU

ReLU activation function should only be used in the hidden layers and Sigmoid should not be used in hidden layers because it makes the model more susceptible to problems during training due to vanishing gradients. Sigmoid is used for binary and multilabel classifications.

ReLU Function

ReLU stands for Rectified Linear Unit. Although it gives an impression of a linear function, ReLU has a derivative function and allows for backpropagation while simultaneously making it computationally efficient. The main catch here is that the ReLU function does not activate all the neurons at the same time. The neurons will only be deactivated if the output of the linear transformation is less than 0.

Sigmoid Function

Sigmoid activation function is one of the most widely used functions for models where we must predict the probability as an output of anything that exists only between the range of 0 and 1. The function is differentiable and provides a smooth gradient, i.e., preventing jumps in output values. This is represented by an S-shape of the sigmoid activation function. It has limitations which include:

- the neurons will only be deactivated if the output of the linear transformation is less than 0.
- as the gradient value approaches zero, the network ceases to learn and suffers from the vanishing gradient problem.

- the output of the logistic function is not symmetric around zero, so the output of all the neurons will be of the same sign. This makes the training of the neural network more difficult and unstable.

Reference: <https://www.v7labs.com/blog/neural-networks-activation-functions#h5>

Output

Model: "sequential_4"

Layer (type)	Output Shape	Param #
dense_12 (Dense)	(None, 80)	3520
dense_13 (Dense)	(None, 30)	2430
dense_14 (Dense)	(None, 1)	31
Total params: 5,981		
Trainable params: 5,981		
Non-trainable params: 0		

Then Neural Network produce a prediction accuracy of 73% which was less than an expected outcome of greater than 75%. The model loss was 56%

```
# Evaluate the model using the test data
model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")
```

268/268 - 1s - loss: 0.5581 - accuracy: 0.7252 - 569ms/epoch - 2ms/step
 Loss: 0.5581002831459045, Accuracy: 0.7252478003501892

Optimization Model

To improve the model performance **NAME** column was not dropped **and** was used along with **CLASSIFICATION** and **APPLICATION_TYPE** for pre-processing.

Increased the hidden layers by adding a third layer and changed the number of neurons in each layer as follows:

- Hidden Layer 1 number of neurons=6
- Hidden Layer 2 number of neurons=13
- Hidden Layer 3 number of neurons=16

Activation function "tanh" (activation="tanh") was used in the first hidden layer and "relu" (activation="relu") activation function was used in the second and third hidden layers.

Activation function "sigmoid" (activation="sigmoid") was used in the output layer.

```
# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
# # Define the model - deep neural net

# Define the model - deep neural net, i.e., the number of input features and hidden nodes for each layer.
number_input_features = len( X_train_scaled[0])
hidden_nodes_layer1=6
hidden_nodes_layer2=13
hidden_nodes_layer3=16
nn = tf.keras.models.Sequential()

# First hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer1, input_dim=number_input_features, activation='tanh'))

# Second hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer2, activation='relu'))

# Third hidden layer
nn.add(tf.keras.layers.Dense(units=hidden_nodes_layer3, activation='relu'))

# Output layer
nn.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))

# Check the structure of the model
nn.summary()
```

Tanh function

Tanh function is very similar to the sigmoid activation function, and even has the same S-shape with the difference in output range of -1 to 1. In Tanh, the larger the input, the closer the output value will be to 1.0, whereas the smaller the input, the closer the output will be to -1.0.

Advantages of using this activation function are:

- The output of the tanh activation function is Zero centred; hence it is easily to map the output values as strongly negative, neutral, or strongly positive.
- Usually used in hidden layers of a neural network as it's values lie between -1 to 1; therefore, the mean for the hidden layer comes out to be 0 or very close to it. It helps in centring the data and makes learning for the next layer much easier.
- Reference: <https://www.v7labs.com/blog/neural-networks-activation-functions#h5>

One of the disadvantages of using this activation function is that of vanishing gradients like the sigmoid activation function.

Output

Model: "sequential_7"

Layer (type)	Output Shape	Param #
dense_27 (Dense)	(None, 6)	2724
dense_28 (Dense)	(None, 13)	91
dense_29 (Dense)	(None, 16)	224
dense_30 (Dense)	(None, 1)	17
Total params: 3,056		
Trainable params: 3,056		
Non-trainable params: 0		

The Neural Network produced a prediction accuracy 79% and a model loss was 45%. Overall there has been an improvement of 6% accuracy and reduction of 10% model loss.

```
✓ 0s # Evaluate the model using the test data
model_loss, model_accuracy = nn.evaluate(X_test_scaled,y_test,verbose=2)
print(f"Loss: {model_loss}, Accuracy: {model_accuracy}")

268/268 - 0s - loss: 0.4503 - accuracy: 0.7915 - 379ms/epoch - 1ms/step
Loss: 0.45026591420173645, Accuracy: 0.7914868593215942
```

Support Vector network (SVM)

SVMs (also support-vector networks) are supervised learning models with associated learning algorithms that analyse data used for classification and regression analysis. SVMs are one of the most robust prediction methods, being based on statistical learning frameworks.

Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.

SVM maps training examples to points in space to maximise the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

A kernel trick to classify nonlinear data without complicated computational expensive mapping our space to a higher dimension for each dataset, can be done by changing the dot product to that of the space that we want and SVM will take care of finding the decision boundary.

References

https://en.wikibooks.org/wiki/Support_Vector_Machines#:~:text=Intuitively%2C%20an%20SVM%20model%20is%20a%20representation%20of,which%20side%20of%20the%20gap%20they%20fall%20on.

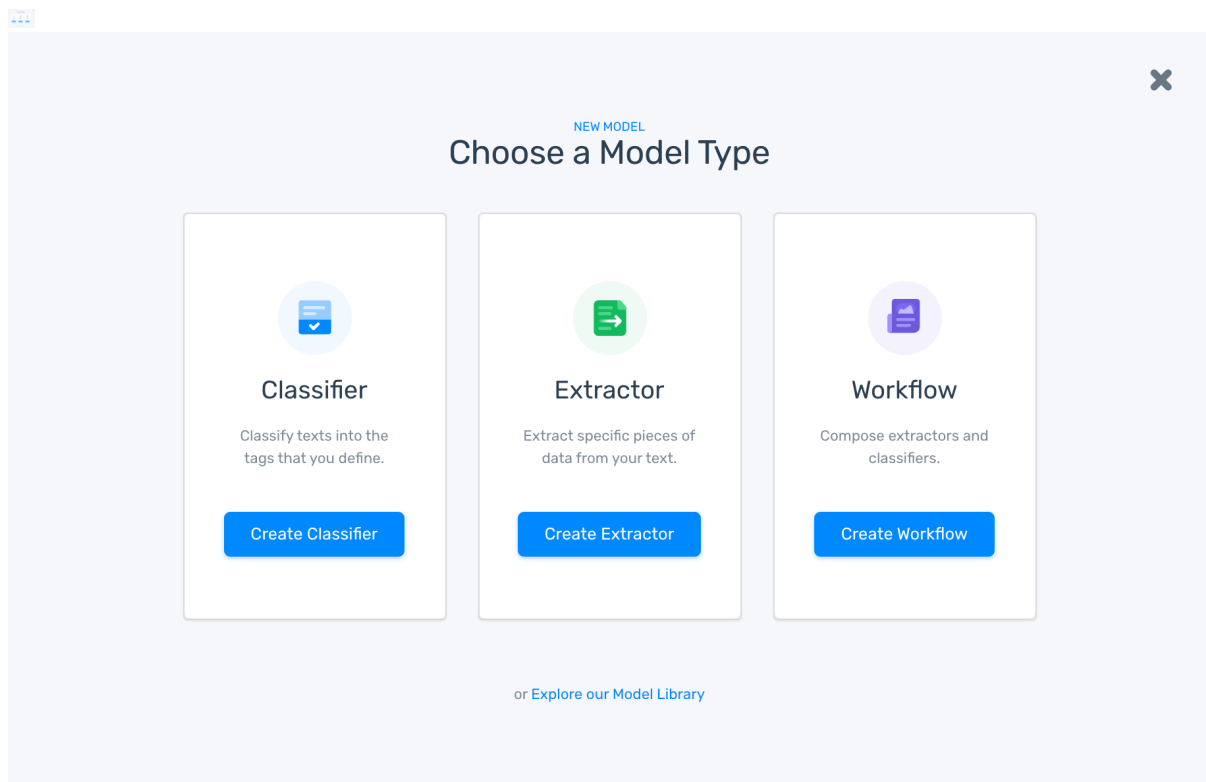
<https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/>

Method

Simple SVM Classifier

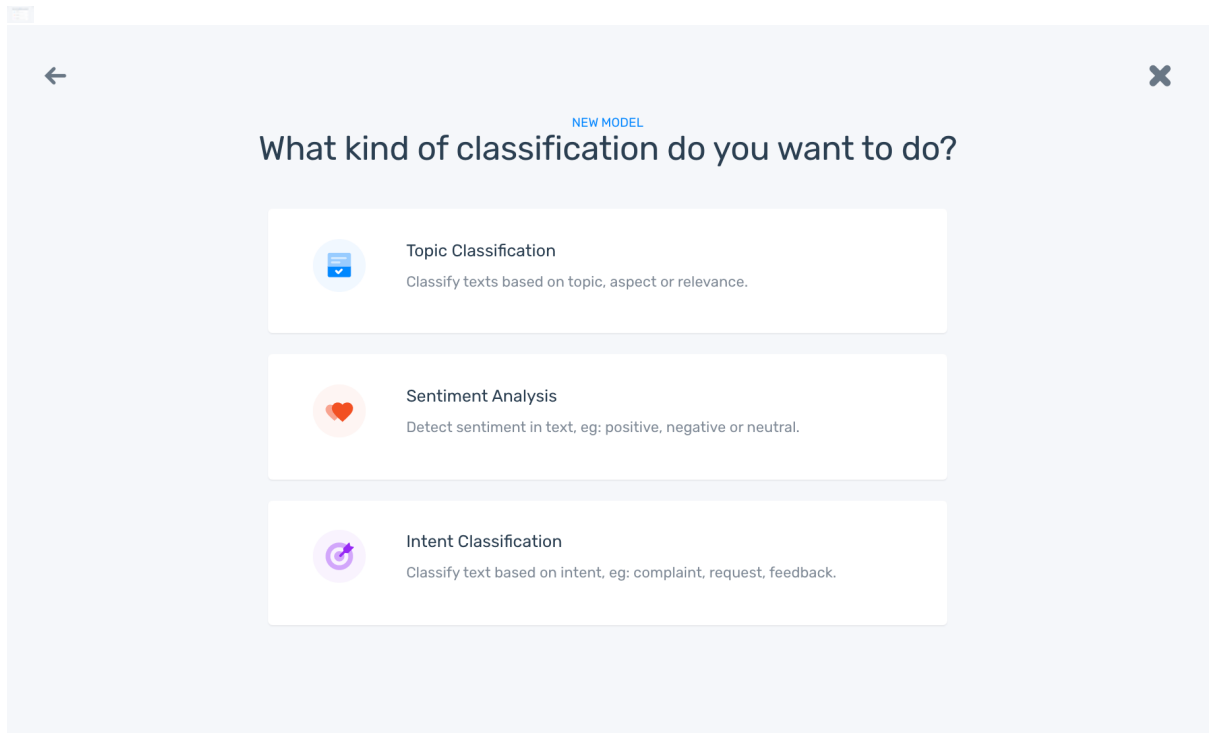
1. Create a new classifier

Go to the [dashboard](#), click on “[Create a Model](#)” and choose “Classifier”.



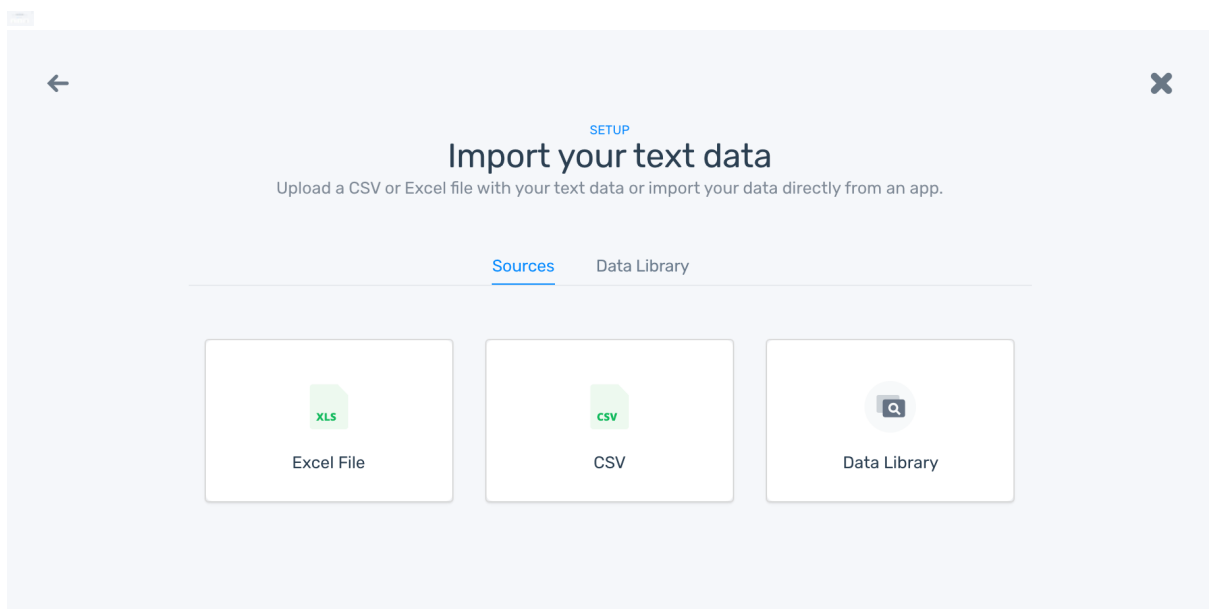
2. Select how you want to classify your data

We're going to opt for a “Topic Classification” model to classify text based on topic, aspect or relevance.



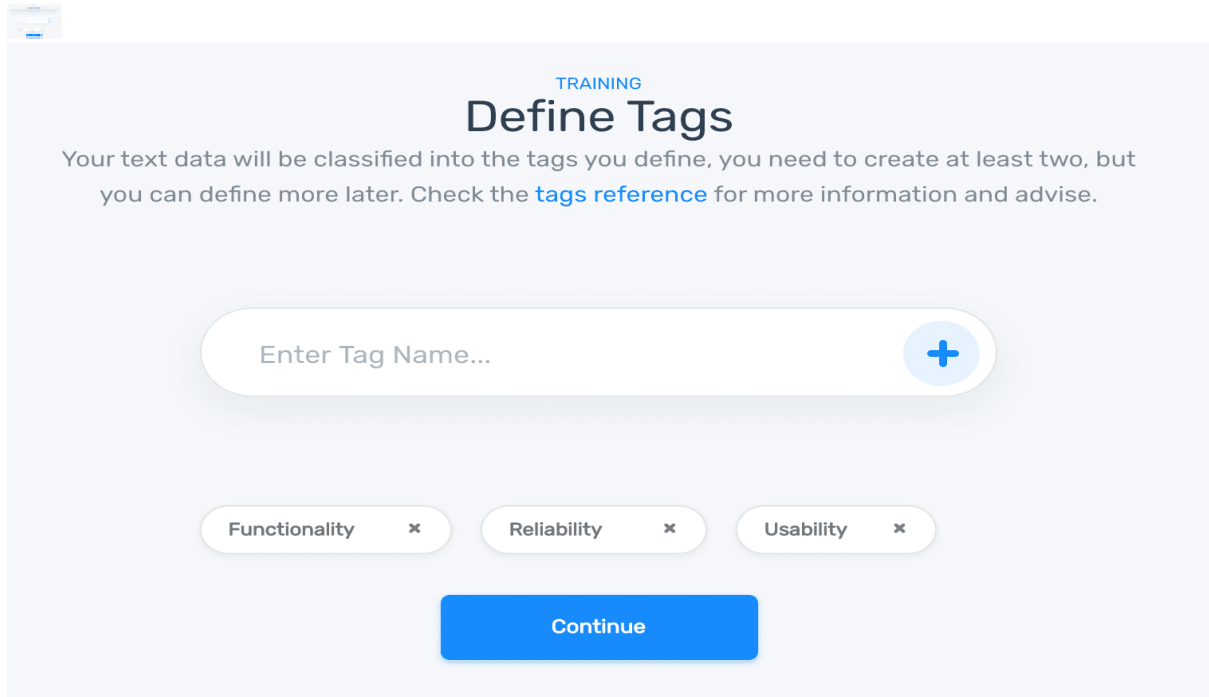
3. Import your training data

Select and upload the data that you will use to train your model. Keep in mind that classifiers learn and get smarter as you feed it more training data. You can import data from CSV or Excel files.



4. Define the tags for your SVM classifier

It's time to define your tags, which you'll use to train your topic classifier. Add at least two tags to get started – you can always add more tags later.



TRAINING

Define Tags

Your text data will be classified into the tags you define, you need to create at least two, but you can define more later. Check the [tags reference](#) for more information and advice.

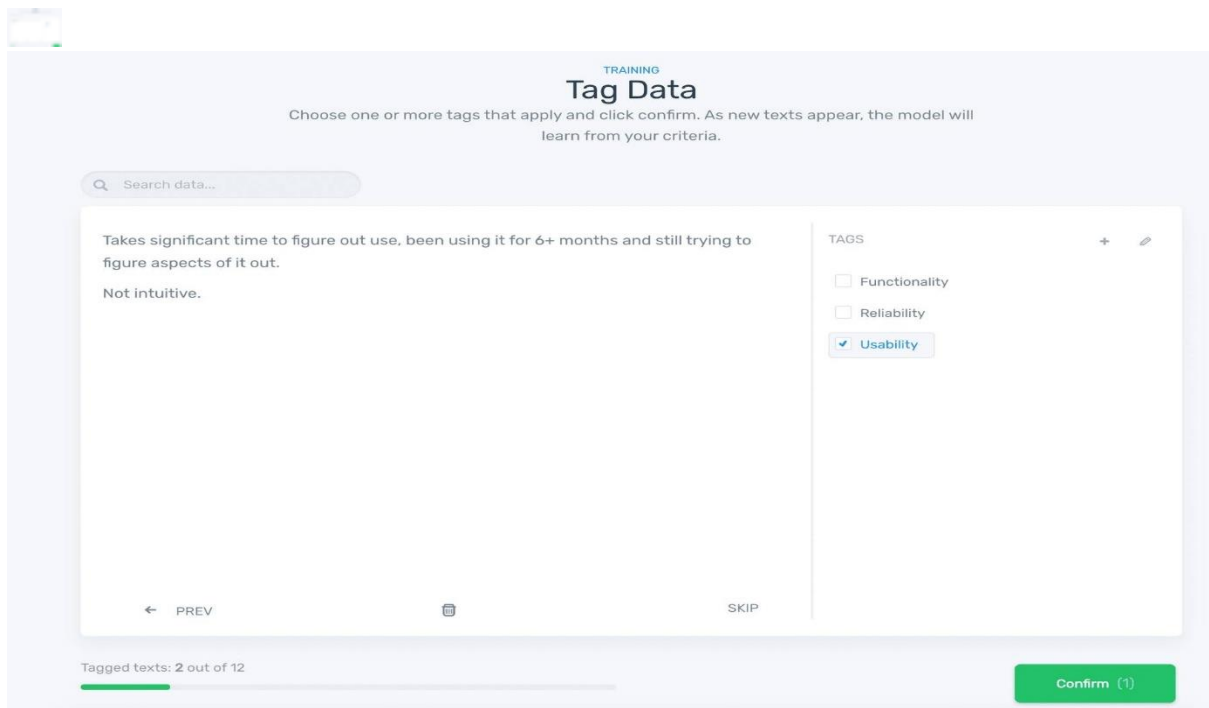
Enter Tag Name... +

Functionality × Reliability × Usability ×

Continue

5. Tag data to train your classifier

Start training your topic classifier by choosing tags for each example:



TRAINING

Tag Data

Choose one or more tags that apply and click confirm. As new texts appear, the model will learn from your criteria.

Search data...

Takes significant time to figure out use, been using it for 6+ months and still trying to figure aspects of it out.
Not intuitive.

Tags

- ☐ Functionality
- ☐ Reliability
- ☒ Usability

← PREV SKIP

Tagged texts: 2 out of 12

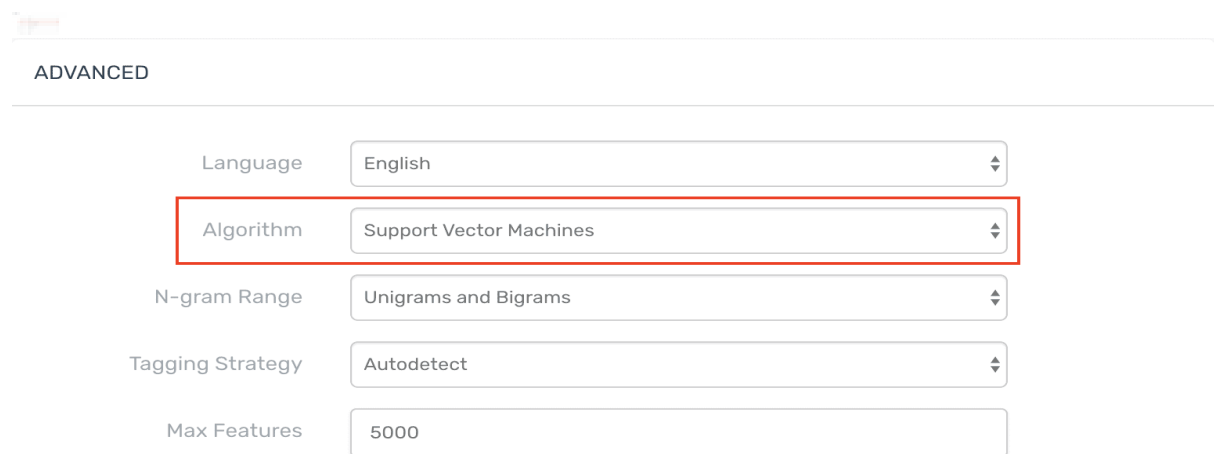
Confirm (1)

After manually tagging some examples, the classifier will start making predictions on its own. If you want your model to be more accurate, you'll have to tag more examples to continue training your model.

The more data you tag, the smarter your model will be.

6. Set your algorithm to SVM

Go to [settings](#) and make sure you select the SVM algorithm in the advanced section.



ADVANCED

Language: English

Algorithm: Support Vector Machines

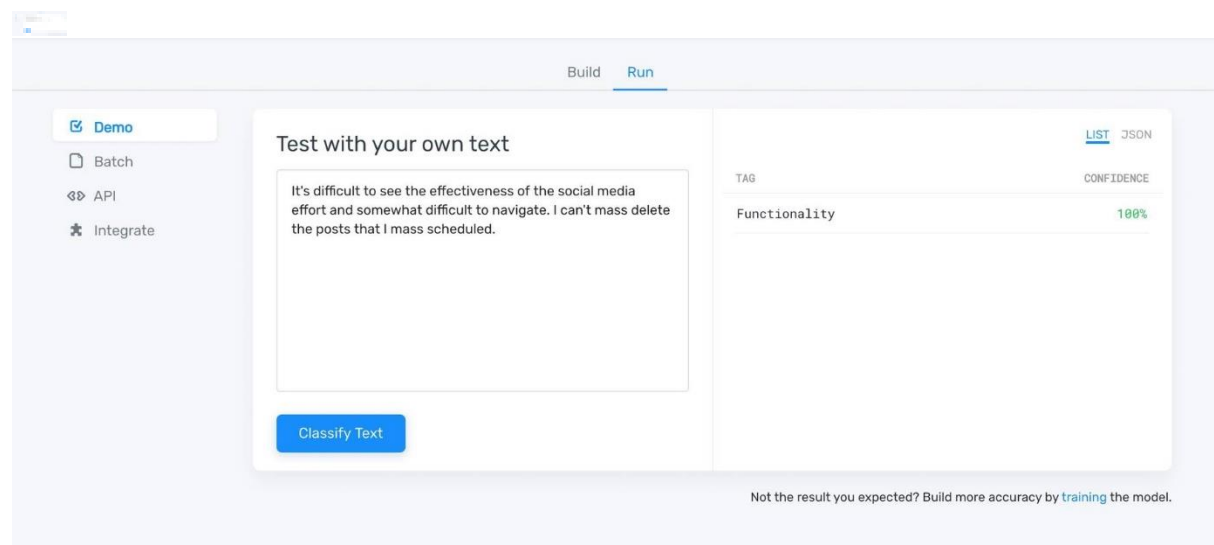
N-gram Range: Unigrams and Bigrams

Tagging Strategy: Autodetect

Max Features: 5000

7. Test Your Classifier

Now you can test your SVM classifier by clicking on “Run” > “Demo”. Write your own text and see how your model classifies the new data:



Build Run

Demo

Batch

API

Integrate

Test with your own text

It's difficult to see the effectiveness of the social media effort and somewhat difficult to navigate. I can't mass delete the posts that I mass scheduled.

Classify Text

TAG	CONFIDENCE
Functionality	100%

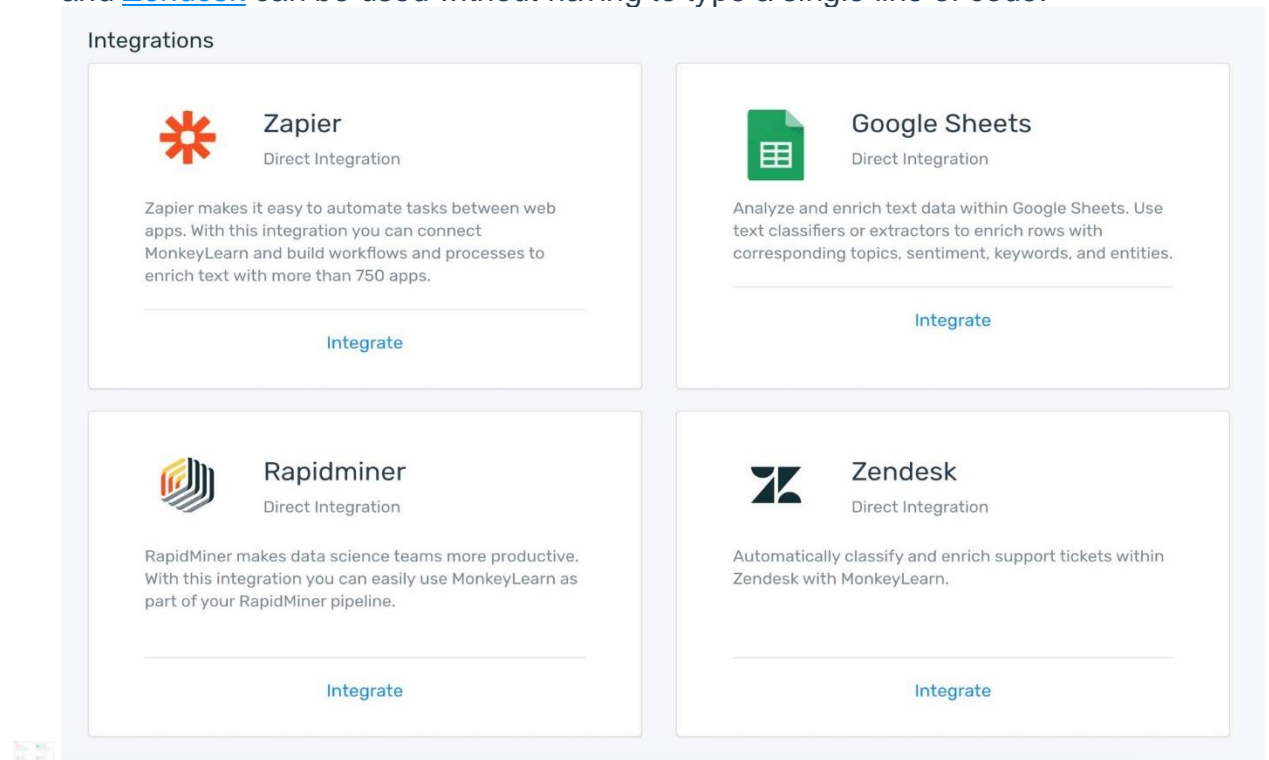
Not the result you expected? Build more accuracy by [training](#) the model.

8. Integrate the topic classifier

You've trained your model to make accurate predictions when classifying text. Now it's time to upload new data! There are three different ways to do this with MonkeyLearn:

1. **Batch processing:** go to “Run” > “Batch” and upload a CSV or Excel file. The classifier will analyze your data and send you a new file with the predictions.
2. **API:** use [MonkeyLearn API](#) to classify new data from anywhere.

3. **Integrations:** connect everyday apps to automatically import new text data into your classifier. Integrations such as [Google Sheets](#), [Zapier](#), and [Zendesk](#) can be used without having to type a single line of code:



And that's the basics of Support Vector Machines!

To sum up:

- A support vector machine allows you to classify data that's linearly separable.
- If it isn't linearly separable, you can use the kernel trick to make it work.
- However, for text classification it's better to just stick to a linear kernel.

With [MLaaS tools](#) like [MonkeyLearn](#), it's extremely simple to implement SVM for text classification and get insights right away.

<https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/>