

I have a Keras model in Python on an input dataset of plaintext. The purpose of the model is to classify text inputs into one of several bins. I seek to use the model to predict the appropriate bin for a manual string input I specify. What is the appropriate function to do this?

Link copied!

If you have already trained a Keras model for text classification, the function you use for new/manual input is simply the model's `predict()` method.

✓ Typical workflow

1. Prepare your input text the same way you prepared training data

This is the most important step.

Whatever preprocessing you used during training must be applied identically:

- Tokenizer (`tokenizer.texts_to_sequences`)
- Vectorizer (e.g., `TextVectorization`)
- Padding (`pad_sequences`)
- Embeddings / normalization
- Etc.

2. Call `model.predict()` on the processed input

Example:

python

 Copy code

```
import numpy as np
```

```
text = ["This is my example text"]
```

```
# text → tokens/ids (must match your training pipeline)
```

```
seq = tokenizer.texts_to_sequences(text)
```

```
padded = pad_sequences(seq, maxlen=MAX_LEN)
```

```
# run prediction
```

```
probs = model.predict(padded)
```

```
# class with highest probability
```

```
predicted_class = np.argmax
```

Link copied!

```
print("Predicted bin:", predicted_class)
```

3. If you have class names, map back:

python

 Copy code

```
bin_name = class_names[predicted_class]
```

```
print(bin_name)
```



If you used TextVectorization

python

 Copy code

```
vec_text = vectorizer(tf.constant([manual_text]))
```

```
prediction = model.predict(vec_text)
```

Summary

To classify **any manual string**, the correct function is:

`model.predict()` , applied to **preprocessed** input identical to training.

If you want, you can show me:

- your model definition,
- your preprocessing pipeline,
- one example input,

and I can give you the exact working code snippet for your setup.



Ok, now I'm trying to load a text dataset from a directory using this code. The error is "No text files found in directory ../manual/. Allowed format: .txt", but I can see two txt files in the directory. Why is this?

Link copied!

This error almost always happens for ~~one or four reasons~~, and they *all* boil down to how `tf.keras.utils.text_dataset_from_directory()` searches for files.

Here are the causes (with fixes):

✓ 1. You are pointing at the *wrong directory level*

`text_dataset_from_directory()` **expects subdirectories** representing class names, and **looks for text files inside those**, not directly in the folder you passed.

Required structure:

bash

 Copy code

```
../manual/  
  classA/  
    file1.txt  
    file2.txt  
  classB/  
    file3.txt
```

✗ Invalid structure:

bash

 Copy code

```
../manual/  
  file1.txt  
  file2.txt
```

If your `.txt` files are directly in `manual/`, the loader will report:

“No text files found in directory ... Allowed format: .txt”

✓ Fix:

Either add subfolders:

bash

 Copy code

```
manual/foo/file1.txt
manual/bar/file2.txt
```

Link copied!

or call the function with `labels=None` and `class_names=None` :

But Keras still *requires subdirectories* even when `labels=None` .

So: **you must use subdirectories.**

✓ 2. Your path is relative to the wrong working directory

Python's working directory may **not** be what you think.

Check:

```
python
```

 Copy code

```
import os
print(os.getcwd())
print(os.listdir("../manual"))
```

If `os.listdir()` fails, you're pointing at the wrong place.

✓ 3. You are on Windows and the filenames include hidden Unicode or spaces

E.g., `"file.txt "` or `"file .txt"` (non-breaking space) will also confuse Keras.

Print filenames:

```
python
```

 Copy code

```
import os
for f in os.listdir("../manual"):
    print(repr(f))
```



4. The files don't exactly

Link copied!

Must be lowercase `.txt` .

Keras won't accept:

- `.TXT`
- `.text`
- `.md`

Check:

python

Copy code

```
import pathlib
for f in pathlib.Path("../manual").glob("*"):
    print(f.suffix)
```



The most common correct solution

Create subdirectories for each label:

css

Copy code

```
manual/
  junk/
    a.txt
  good/
    b.txt
```

Then:

python

Copy code

```
ds = tf.keras.utils.text_dataset_from_directory(
    "../manual",
    batch_size=32
)
```

If you want, send me

Link copied!

- your exact directory structure (`tree ../manual`)
- your exact code snippet

and I'll tell you the exact reason in 10 seconds.



keras-attempt2.ipynb
File

Ok, here's my current working notebook - with everything working except the final prediction. Can you explain to me what I'm doing wrong?

You're *very* close — the only issue is that in your last two lines you are calling the **wrong model**.

In your notebook you have **two models**:

1. `model` – the model *before* you wrapped text-vectorization into it
2. `end_to_end_model` – the model that accepts **raw strings** and does:

csharp

Copy code

raw **string** → `vectorize_layer` → `model` → prediction

But your final cell does this:

python

Copy code

```
vec_text = vectorize_text(tf.constant(["hello world"]), 0)
model.predict(vec_text)
```

This is wrong for two reasons:

❌ 1. You're predicting with `model.predict(text)`, which expects already-vectorized inputs

Link copied!

`model` expects a **padded integer sequence**, not a tuple `(vectorized_text, label)`.

Your `vectorize_text()` function returns a **pair**:

python

 Copy code

```
(vectorize_layer(text), label)
```

So `model.predict(vec_text)` is actually feeding a **tuple**, not just the vectorized text.

❌ 2. Even if you extracted the vector, the *correct* model for raw text is `end_to_end_model`

Only `end_to_end_model` has the vectorizer built into its input pipeline.

✅ Correct ways to predict

★ Easiest (since you built an end-to-end model)

Just pass a raw string to the end-to-end model:

python

 Copy code

```
end_to_end_model.predict(["hello world"])
```

That's *exactly* what the model was designed for.

If you prefer manual vectorization:

python

 Copy code

```
vec = vectorize_layer(tf.constant(["hello world"]))  
model.predict(vec)
```

Note: **don't call your** `vectorize_text` helper

That helper is for mapping over `tf.constant` and always returns (x, y) pairs.

Link copied!

Summary of what to change

Replace:

python

 Copy code

```
vec_text = vectorize_text(tf.constant(["hello world"]), 0)
model.predict(vec_text)
```

with:

python

 Copy code

```
end_to_end_model.predict(["hello world"])
```

That will work correctly.

If you want, I can rewrite the last notebook cell into the cleanest working form.

     

When I run `end_to_end_model.predict(["hello world"])` I get -----

ValueError Traceback (most recent call last)

Cell In[20], line 1

----> 1 end_to_end_model.predict(["hello world"])

File ~/pyenv/versions/3.11.8/lib/python3.11/site-packages/keras/src/utils/traceback_utils.py:122, in filter_traceback.<locals>.error_handler(*args, **kwargs)

119 filtered_tb =

```

__process_traceback_frames(e.__traceback__)
121     # stack trace, call:
122     #
keras.config.disable_traceback_filtering()
--> 122     raise e.with_traceback(filtered_tb) from None
123 finally:
124     del filtered_tb

File ~/pyenv/versions/3.11.8/lib/python3.11/site-
packages/keras/src/trainers/data_adapters/__init__.py:16
3, in get_data_adapter(x, y, sample_weight, batch_size,
steps_per_epoch, shuffle, class_weight)
155     return GeneratorDataAdapter(x)
156     # TODO: should we warn or not?
157     # warnings.warn(
158     #     "shuffle=True was passed, but will be
ignored since the "
(...)
161     # )
162 else:
--> 163     raise ValueError(f"Unrecognized data type: x=
{x} (of type {type(x)})")

ValueError: Unrecognized data type: x=['hello world'] (of
type <class 'list'>)

```

That error means **your end-to-end model is NOT expecting a string input**, even though you intended it to.

Keras only accepts raw Python lists of strings when the model's *first layer has dtype `tf.string`*.

Your error:

powershell

 Copy code

```
ValueError: Unrecognized data type: x=['hello world']
```

means the model is expecting **numeric tensors**, not strings.

This happens if the way you created `vectorize_layer` caused the vectorizer to **freeze to numeric mode** instead

Link copied!

✅ Try this first (usually fixes it)

Use either `tf.constant()` or a NumPy array instead of a Python list:

python

Copy code

```
end_to_end_model.predict(tf.constant(["hello world"]))
```

or

python

Copy code

```
import numpy as np
end_to_end_model.predict(np.array(["hello world"]))
```

If **this works**, the model is string-aware internally.

If that STILL fails →

Your `end_to_end_model` was not constructed as:

python

Copy code

```
inputs = tf.keras.Input(shape=(1,), dtype=tf.string)
x = vectorize_layer(inputs)
outputs = model(x)
end_to_end_model = tf.keras.Model(inputs, outputs)
```

Instead, you probably used something like:

python

Copy code

```
end_to_end_model = tf.keras.Sequential([vectorize_layer, model])
```

which **breaks string input** unless you explicitly set:

python

Copy code

```
tf.keras.Input(dtype=tf.st
```

Link copied!

✅ Fix: Rebuild the end-to-end model correctly

Here is the correct construction:

python

 Copy code

```
string_input = tf.keras.Input(shape=(1,), dtype=tf.string)
x = vectorize_layer(string_input)
preds = model(x)
end_to_end_model = tf.keras.Model(string_input, preds)
```

Now prediction works:

python

 Copy code

```
end_to_end_model.predict(["hello world"])
```

★ The real issue (guaranteed):

Your `end_to_end_model` is currently expecting **ints**, not **strings**, because its first layer is not defined with `dtype=tf.string`.

Rebuilding with an explicit string input fixes it permanently.

If you want, upload the last 20 lines of your notebook where you define `end_to_end_model`

I'll tell you exactly which line is causing the problem and give you the minimal corrected version.

     ...