Data pre-processing

RECURRENT NEURAL NETWORKS FOR LANGUAGE MODELING IN PYTHON

David Cecchini
Data Scientist

datacamp

Text classification

Applications of text classification:

- Automatic news classification
- Document classification for businesses
- Queue segmentation for customer support
- Many more!

Changes from binary classification

What change from binary to multi class:

- Shape of the output variable `y`
- Number of units on the output layer
- Activation function on the output layer
- Loss function

Changes from binary classification

Shape of the output variable `y` :

- One-hot encoding of the classes

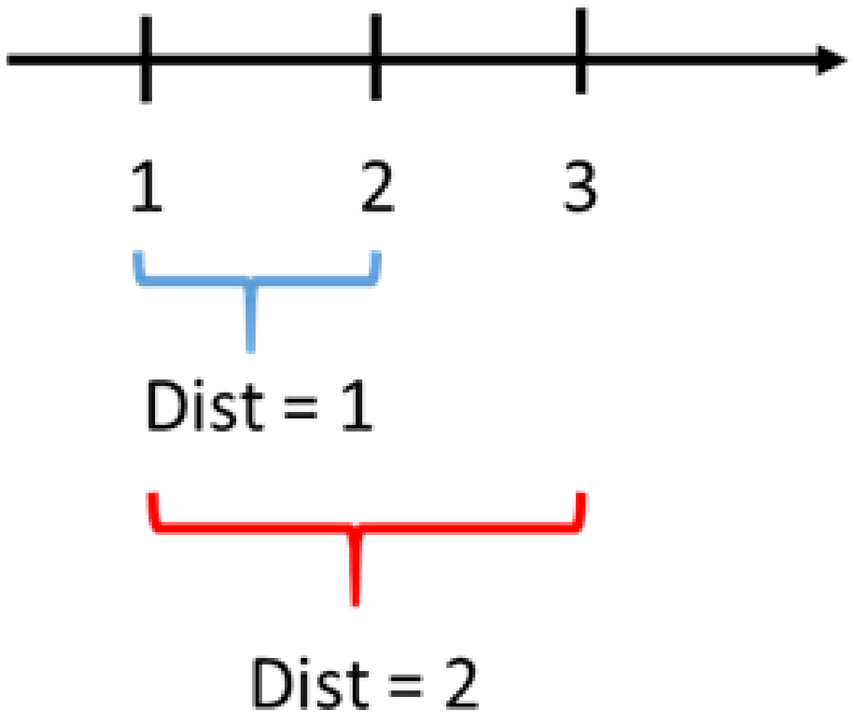
```
# Example: num_classes = 3
y[0] = [0, 1, 0]
y.shape = (N, num_classes)
```

Number of units on the output layer:

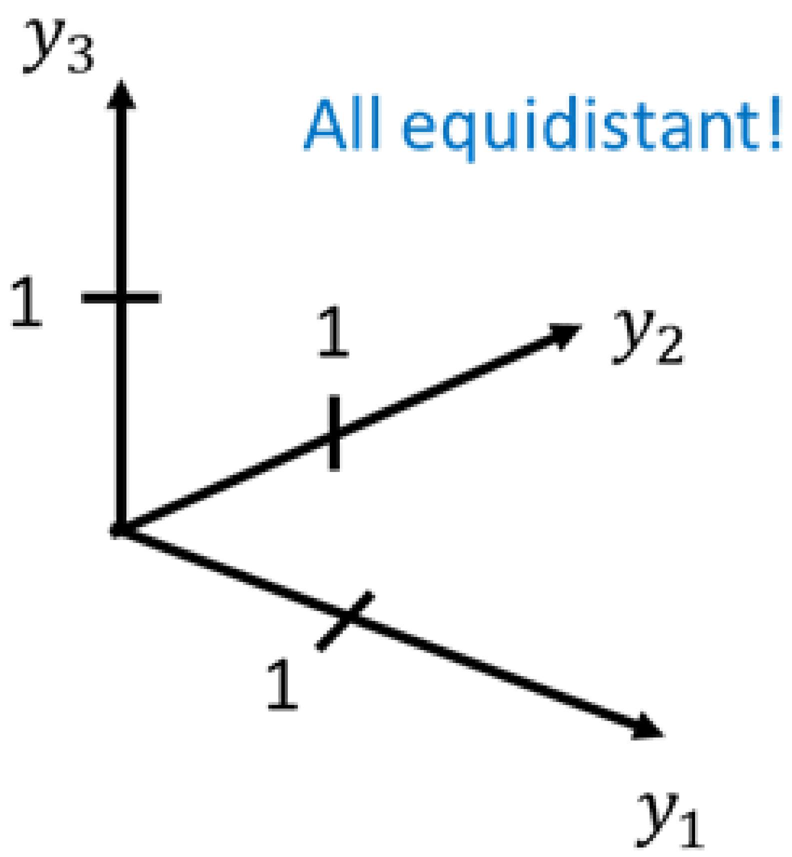
```
# Output layer
model.add(Dense(num_classes))
```

Changes from binary classification

$y \in \mathbb{R}$



$y \in \mathbb{R}^3$



Changes from binary classification

Activation function on the output layer:

- `softmax` gives the probability of every class

```
# Output layer
model.add(Dense(num_classes, activation="softmax"))
```

Loss function:

- Instead of binary, we use categorical cross-entropy

```
# Compile the model
model.compile(loss='categorical_crossentropy')
```

Preparing text categories for keras

```
y = ["sports", "economy", "data_science", "sports", "finance"]
# Transform to pandas series object
y_series = pd.Series(y, dtype="category")
# Print the category codes
print(y_series.cat.codes)
```

```
0    3
1    1
2    0
3    3
4    2
```

Pre-processing y

```
from keras.utils.np_utils import to_categorical
y = np.array([0, 1, 2])

# Change to categorical
y_prep = to_categorical(y)
print(y_prep)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

Let's practice!

RECURRENT NEURAL NETWORKS FOR LANGUAGE MODELING IN PYTHON

Transfer learning for language models

RECURRENT NEURAL NETWORKS FOR LANGUAGE MODELING IN PYTHON

David Cecchini
Data Scientist

datacamp

The idea behind transfer learning

Transfer learning:

- Start with better than random initial weights
- Use models trained on very big datasets
- "Open-source" data science models

Available architectures

Base example: `I really loved this movie`

- Word2Vec
  - Continuous Bag of Words (CBOW) `X = [I, really, this, movie], y = loved`
  - Skip-gram `X = loved, y = [I, really, this, movie]`
- FastText `X = [I, rea, eal, all, lly, really, ...], y = loved`
  - Uses words and n-grams of chars
- ELMo `X = [I, really, loved, this], y = movie`
  - Uses words, embeddings per context
  - Uses Deep bidirectional language models (biLM)
- Word2Vec and FastText are available on package `gensim` and ELMo on `tensorflow_hub`

Example using Word2Vec

```
from gensim.models import word2vec
# Train the model
w2v_model = word2vec.Word2Vec(tokenized_corpus, size=embedding_dim,
                              window=neighbor_words_num, iter=100)
# Get top 3 similar words to "captain"
w2v_model.wv.most_similar(["captain"], topn=3)
```

```
[('sweatpants', 0.7249663472175598),
('kirk', 0.7083336114883423),
('larry', 0.6495886445045471)]
```

Example using FastText

```
from gensim.models import fasttext
# Instantiate the model
ft_model = fasttext.FastText(size=embedding_dim, window=neighbor_words_num)
# Build vocabulary
ft_model.build_vocab(sentences=tokenized_corpus)
# Train the model
ft_model.train(sentences=tokenized_corpus,
               total_examples=len(tokenized_corpus),
               epochs=100)
```

Let's practice!

RECURRENT NEURAL NETWORKS FOR LANGUAGE MODELING IN PYTHON

Multi-class classification models

RECURRENT NEURAL NETWORKS FOR LANGUAGE MODELING IN PYTHON

David Cecchini
Data Scientist

datacamp

Review of the Sentiment classification model

```
# Build and compile the model
model = Sequential()
model.add(Embedding(10000, 128))
model.add(LSTM(128, dropout=0.2))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Model architecture

Same architecture can be used

```
# Build the model
model = Sequential()
model.add(Embedding(10000, 128))
model.add(LSTM(128, dropout=0.2))
# Output layer has `num_classes` units and uses `softmax`
model.add(Dense(num_classes, activation="softmax"))
# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
...
```

20 News Group dataset

20 News Groups Dataset

- Available on `sklearn.datasets import fetch_20newsgroups`

```
# Import the function to load the data
from sklearn.datasets import fetch_20newsgroups
# Download train and test sets
news_train = fetch_20newsgroups(subset='train')
news_test = fetch_20newsgroups(subset='test')
```

20 News Group dataset

The data has the following attributes:

- `news_train.DESCR` : Documentation.
- `news_train.data` : Text data.
- `news_train.filenames` : Path to the files on disk.
- `news_train.target` : Numerical index of the classes.
- `news_train.target_names` : Unique names of the classes.

Pre-process text data

```python
# Import modules
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils.np_utils import to_categorical
# Create and fit the tokenizer
tokenizer = Tokenizer()
tokenizer.fit_on_texts(news_train.data)
# Create the (X, Y) variables
X_train = tokenizer.texts_to_sequences(news_train.data)
X_train = pad_sequences(X_train, maxlen=400)
Y_train = to_categorical(news_train.target)
```

Training on data

Train the model on training data

```
# Train the model
model.fit(X_train, Y_train,
          batch_size=64, epochs=100)


# Evaluate on test data
model.evaluate(X_test, Y_test)
```

Let's practice!

RECURRENT NEURAL NETWORKS FOR LANGUAGE MODELING IN PYTHON

Assessing the model's performance

RECURRENT NEURAL NETWORKS FOR LANGUAGE MODELING IN PYTHON

David Cecchini
Data Scientist

datacamp

Accuracy is not too informative

20 classes task with 80% accuracy. Is the model good?

- Can it classify all the classes correctly?
- Is the accuracy the same for each class?
- Is the model overfitting on the majority class?

I have no idea!

Confusion matrix

Checking true and predicted for each class

		Predicted		
		sci.space	alt.atheism	soc.religion.christian
True class	sci.space	76	2	0
	alt.atheism	7	1	2
	soc.religion.christian	9	0	3

Precision

Precision

$$\text{Precision}_{\text{class}} = \frac{\text{Correct}_{\text{class}}}{\text{Predicted}_{\text{class}}}$$

In the example:

$$\text{Precision}_{\text{sci.space}} = \frac{76}{76+7+9} = 0.83$$

$$\text{Precision}_{\text{alt.atheism}} = \frac{1}{2+1+0} = 0.33$$

$$\text{Precision}_{\text{soc.religion.christian}} = \frac{3}{0+2+3} = 0.60$$

Recall

Recall

$$\text{Recall}_{\text{class}} = \frac{\text{Correct}_{class}}{N_{\text{class}}}$$

In the example:

$$\text{Recall}_{\text{sci.space}} = \frac{76}{76+2+0} = 0.97$$

$$\text{Recall}_{\text{alt.atheism}} = \frac{1}{7+1+2} = 0.10$$

$$\text{Recall}_{\text{soc.religion.christian}} = \frac{3}{9+0+3} = 0.25$$

F1-Score

F1-Score

$$\text{F1 score} = 2 * \frac{\text{precision}_{\text{class}} * \text{recall}_{\text{class}}}{\text{precision}_{\text{class}} + \text{recall}_{\text{class}}}$$

In the example:

$$f1score_{sci.space} = 2\frac{0.83 * 0.97}{0.83 + 0.97} = 0.89$$

$$f1score_{alt.atheism} = 2\frac{033 * 0.10}{033 + 0.10} = 0.15$$

$$f1score_{soc.religion.christian} = 2\frac{060 * 0.25}{060 + 0.25} = 0.35$$

Sklearn confusion matrix

```
from sklearn.metrics import confusion_matrix
# Build the confusion matrix
confusion_matrix(y_true, y_pred)
```

Output:

```
array([[76,  2,  0],
       [ 7,  1,  2],
       [ 9,  0,  3]], dtype=int64)
```

Performance metrics

Metrics from sklearn

```
# Functions of sklearn
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
```

Performance metrics

```
# Accuracy
print(accuracy_score(y_true, y_pred))
```

```
$ 0.80
```

Add `average=None` to precison, recall and f1 score functions

```
print(precision_score(y_true, y_pred, average=None))
print(recall_score(y_true, y_pred, average=None))
print(f1_score(y_true, y_pred, average=None))
```

```
$ array([0.83, 0.33, 0.60])
$ array([0.97, 0.10, 0.25])
$ array([0.89, 0.15, 0.35])
```

Classification report

One function measure all:

```
lab_names = ['sci.space', 'alt.atheism', 'soc.religion.christian']
print(classification_report(y_true, y_pred, target_names=lab_names))
```

```
                        precision    recall  f1-score   support


             sci.space       0.83      0.97      0.89        78
           alt.atheism       0.33      0.10      0.15        10
soc.religion.christian       0.60      0.25      0.35        12


             micro avg       0.80      0.80      0.80       100
             macro avg       0.59      0.44      0.47       100
          weighted avg       0.75      0.80      0.76       100
```

Let's practice!

RECURRENT NEURAL NETWORKS FOR LANGUAGE MODELING IN PYTHON