Classification using vectored text

성균관대학교
정윤경

Classification with Machine Learning algorithms using sklearn

Dataset: the 20 Newsgroups data set

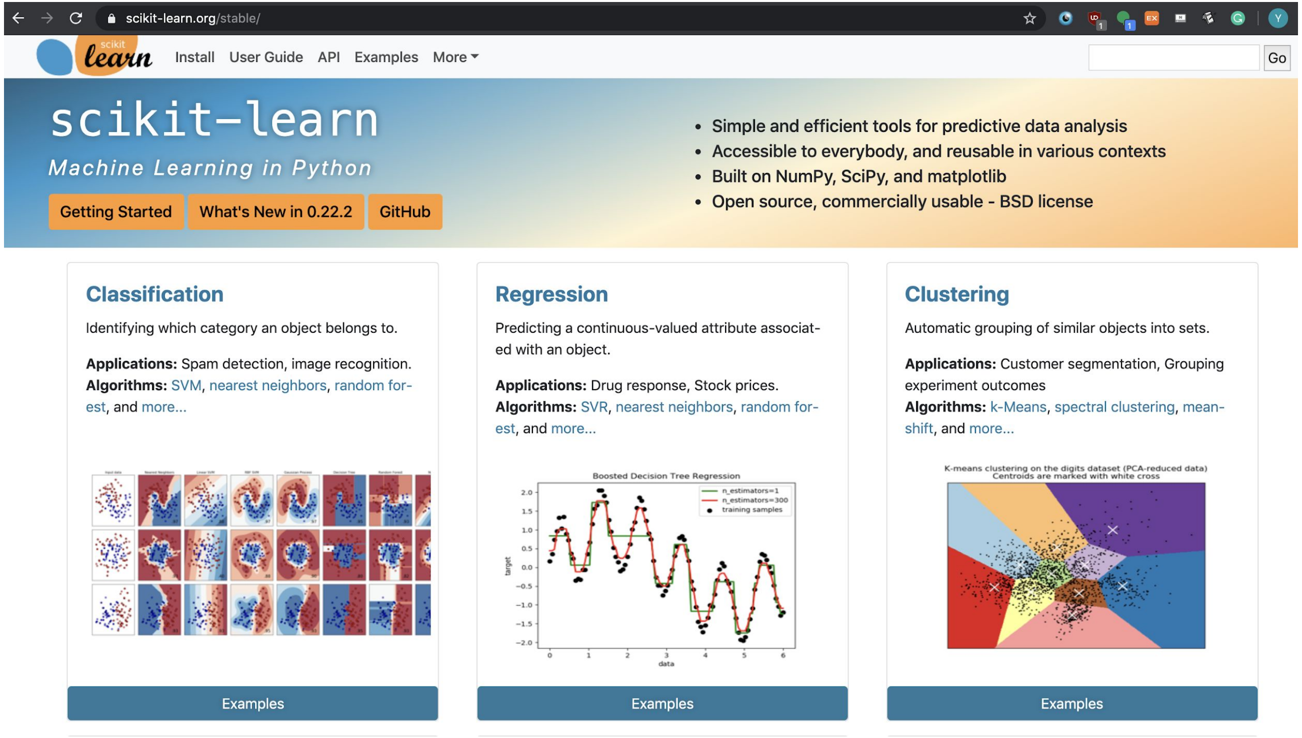
- Source: http://qwone.com/~jason/20Newsgroups/
- The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups.
- It was originally collected by Ken Lang, probably for his *Newsweeder: Learning to filter netnews* paper, though he does not explicitly mention this collection
- The 20 newsgroups collection has become a popular data set for experiments in text applications of machine learning techniques, such as text classification and text clustering

Dataset: organization

- The data is organized into 20 different newsgroups, each corresponding to a different topic. Some of the newsgroups are very closely related to each other (e.g. comp.sys.ibm.pc.hardware / comp.sys.mac.hardware), while others are highly unrelated (e.g misc.forsale / soc.religion.christian). Here is a list of the 20 newsgroups, partitioned (more or less) according to subject matter:

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

Scikit-learn (https://scikit-learn.org/)



Classification with the scikit-learn package

- https://scikit-learn.org/0.19/datasets/twenty_newsgroups.html

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
categories = ['alt.atheism', 'talk.religion.misc','comp.graphics', 'sci.space']
newsgroups_train = fetch_20newsgroups(subset='train', categories=categories)
vectorizer = TfidfVectorizer()
vectors = vectorizer.fit_transform(newsgroups_train.data)

from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
newsgroups_test = fetch_20newsgroups(subset='test', categories=categories)
vectors_test = vectorizer.transform(newsgroups_test.data)
clf = MultinomialNB(alpha=.01)
clf.fit(vectors, newsgroups_train.target)
pred = clf.predict(vectors_test)
metrics.f1_score(newsgroups_test.target, pred, average='macro')
>>> 0.88213592402729568
```

Overfitting problem

- Almost every group is distinguished by whether headers such as NNTP-Posting-Host: and Distribution: appear more or less often.
- Another significant feature involves whether the sender is affiliated with a university, as indicated either by their headers or their signature.
- The word “article” is a significant feature, based on how often people quote previous posts like this: “In article [article ID], [name] <[e-mail address]> wrote:”
- Other features match the names and e-mail addresses of particular people who were posting at the time

```
# extract the most informative features
import numpy as np
def show_top10(classifier, vectorizer, categories):
    feature_names = np.asarray(vectorizer.get_feature_names())
    for i, category in enumerate(categories):
        top10 = np.argsort(classifier.coef_[i])[-10:]
        print("%s: %s" % (category, " ".join(feature_names[top10])))


>>> show_top10(clf, vectorizer, newsgroups_train.target_names)
alt.atheism: sgi livesey atheists writes people caltech com god keith edu
comp.graphics: organization thanks files subject com image lines
university edu graphics
sci.space: toronto moon gov com alaska access henry nasa edu space
talk.religion.misc: article writes kent people christian jesus sandvik edu
com
```

Filtering text

- With such an abundance of clues that distinguish newsgroups, the classifiers barely have to identify topics from text at all, and they all perform at the same high level
- For this reason, the functions that load 20 Newsgroups data provide a parameter called **remove**, telling it what kinds of information to strip out of each file. remove should be a tuple containing any subset of ('headers', 'footers', 'quotes'), telling it to remove headers, signature blocks, and quotation blocks respectively

```
# remove metadata
newsgroups_test = fetch_20newsgroups(subset='test',
                                     remove=('headers', 'footers', 'quotes'),
                                     categories=categories)
vectors_test = vectorizer.transform(newsgroups_test.data)

newsgroups_train = fetch_20newsgroups(subset='train',
                                      remove=('headers', 'footers', 'quotes'),
                                      categories=categories)
vectors = vectorizer.fit_transform(newsgroups_train.data)
clf = MultinomialNB(alpha=.01)
clf.fit(vectors, newsgroups_train.target)
vectors_test = vectorizer.transform(newsgroups_test.data)
pred = clf.predict(vectors_test)

>>> metrics.f1_score(newsgroups_test.target, pred, average='macro')
0.7699517518452172
```

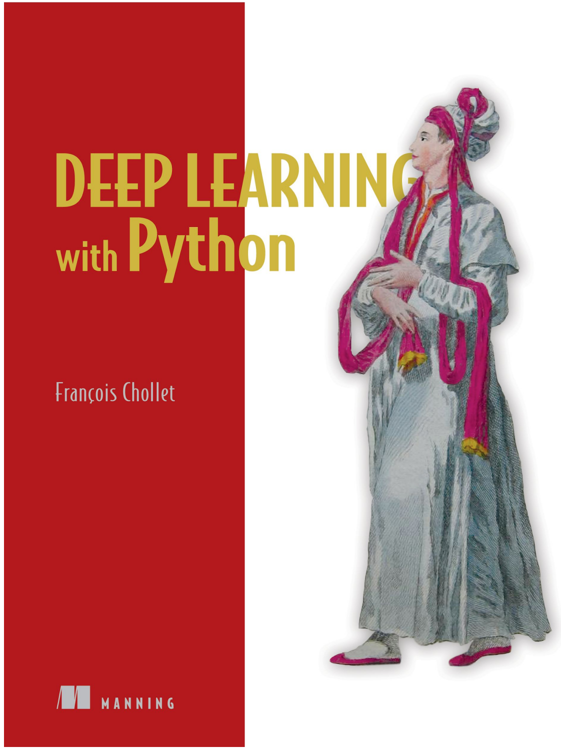
Further analysis

- Experiment with different machine learning algorithms
  - SVM, KNN, logistic regression varying the parameter settings
  - Run Decision Tree algorithm to find rules that classify the news
- Experiment with simple MLP algorithms
  - Design the network structure, hidden layers and nodes
  - Vary the training parameter setting such as activation function, early stopping, iterations, etc.
  - Vary the evaluation setting such as loss function

Binary Classification using Densely-connected Neural Network

https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/first_edition/3.5-classifying-movie-reviews.ipynb

Examples from



Binary Classification: Movie Reviews

- Dataset contains 50,000 highly-polarized reviews from IMDB (about 80MB)
- Split into 25,000 reviews for training and 25,000 reviews for testing, each set consisting in 50% negative and 50% positive reviews
- Represent each review as a list of word indices
- Keep the top 10,000 most frequently occurring words in the training data

```
%tensorflow_version 1.x
from keras.datasets import imdb
(train_data, train_labels), (test_data, test_labels) =
imdb.load_data(num_words=10000)
```

"? this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you could just imagine being there robert ? is an amazing actor and now the same being director ? father came from the same scottish island as myself so i loved the fact there was a real connection with this film the witty remarks throughout the film were great it was just brilliant so much that i bought the film as soon as it was released for ? and would recommend it to everyone to watch and the fly fishing was amazing really cried at the end it was so sad and you know what they say if you cry at a film it must have been good and this definitely was also ? to the two little boy's that played the ? of norman and paul they were just brilliant children are often left out of the ? list i think because the stars that play them all grown up are such a big profile for the whole film but these children are amazing and should be praised for what they have done don't you think the whole story was so lovely because it was true and was someone's life after all that was shared with us all"

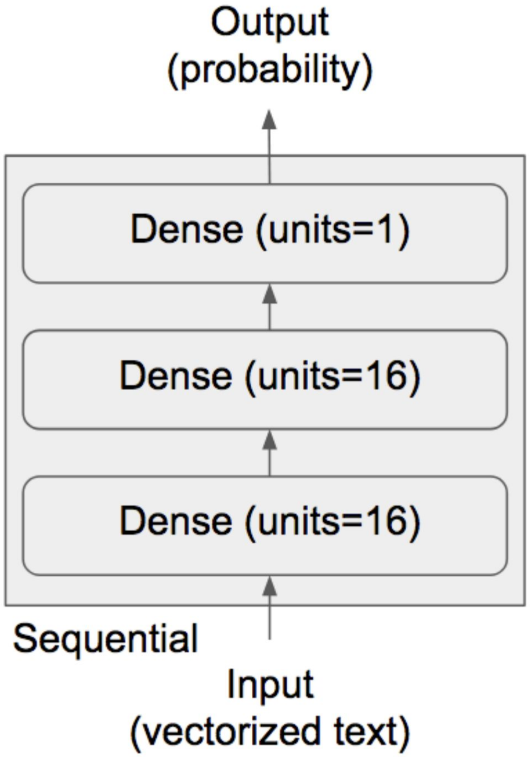
Vectorization

- Use one-hot encoding scheme using the most frequent 10,000 words
- For labels, 0 stands for "negative" and 1 stands for "positive"

```
import numpy as np
def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.  # set specific indices of results[i] to 1s
    return results

# vectorized training/test data
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)

#  train_labels and test_labels are lists of 0s and 1s
y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

Modeling

- Two intermediate layers with 16 hidden units each, and a third layer which will output the scalar prediction regarding the sentiment of the current review



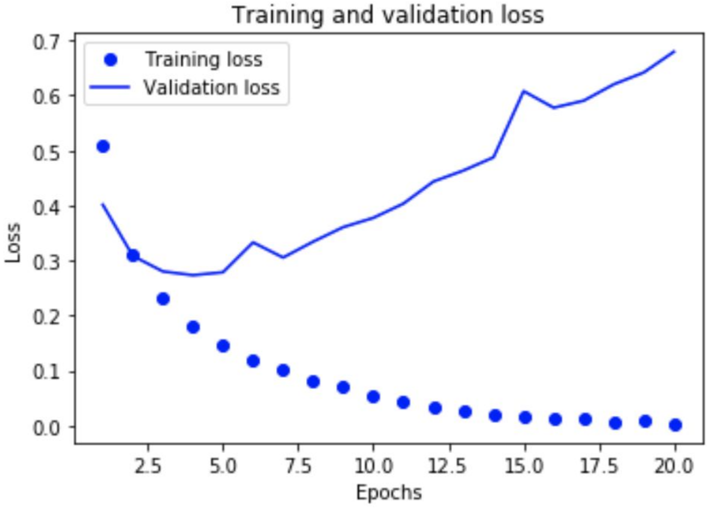
Modeling

- Two intermediate layers with 16 hidden units each, and a third layer which will output the scalar prediction regarding the sentiment of the current review
- The intermediate layers use **relu** as their "activation function", and the final layer use a **sigmoid** activation so as to output a probability (a score between 0 and 1, indicating how likely the sample is to have the target "1", i.e. how likely the review is to be positive)

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy', metrics=['accuracy'])

x_val = x_train[:10000]
partial_x_train = x_train[10000:]

y_val = y_train[:10000]
partial_y_train = y_train[10000:]

history = model.fit(partial_x_train,   partial_y_train,      epochs=20,
                    batch_size=512,      validation_data=(x_val, y_val))
```

Evaluation

- achieves an accuracy of 88%. With state-of-the-art approaches, one should be able to get close to 95%




```
results = model.evaluate(x_test, y_test)
print(results) # loss, acc [0.29184698499679568, 0.88495999999999997]

# visualization
history_dict = history.history
history_dict.keys()

import matplotlib.pyplot as plt
acc = history.history['acc']  # binary_accuracy
val_acc = history.history['val_acc'] # val_binary_accuracy
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)
plt.plot(epochs, loss, 'bo', label='Training loss') # "bo" is "blue dot"
plt.plot(epochs, val_loss, 'b', label='Validation loss') # b is "solid blue line"
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Conclusion

- There's usually quite a bit of **preprocessing** you need to do on your raw data in order to be able to feed it -- as tensors -- into a neural network
- Stacks of Dense layers with **relu** activations can solve a wide range of problems (including sentiment classification)
- In a binary classification problem, your **network should end with a Dense layer with 1 unit and a sigmoid activation**, i.e. the output of your network should be a scalar between 0 and 1, encoding a probability.
- With such a scalar sigmoid output, on a binary classification problem, the **loss function you should use is binary_crossentropy**
- The **rmsprop** optimizer is generally a good enough choice of optimizer, whatever your problem
- As they get better on their training data, neural networks eventually start **overfitting** and end up obtaining increasingly worse results on data never-seen-before. Make sure to always monitor performance on data that is outside of the training set

Multinomial Classification using Densely-connected Neural Network

https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/first_edition/3.6-classifying-newswires.ipynb

Dataset: Reuters newswires

- a set of short newswires and 46 different mutually-exclusive topics, published by Reuters in 1986
- single-label, multi-class classification
- some topics are more represented than others, but each topic has at least 10 examples in the training set
- each example is a list of integers (word indices)

```
import keras #2.0 version
from keras.datasets import reuters

(train_data, train_labels), (test_data, test_labels) =
reuters.load_data(num_words=10000)

len(train_data)        # 8982
len(test_data)         # 2246
```

'? ? ? said as a result of its december acquisition of space co it expects earnings per share in 1987 of 1 15 to 1 30 dlrs per share up from 70 cts in 1986 the company said pretax net should rise to nine to 10 mln dlrs from six mln dlrs in 1986 and rental operation revenues to 19 to 22 mln dlrs from 12 5 mln dlrs it said cash flow per share this year should be 2 50 to three dlrs reuter 3'

Vectorization

- To vectorize the **labels**, there are two possibilities: we could just cast the label list as an integer tensor, or we could use a "one-hot" encoding, also called "categorical encoding"

```
from keras.utils.np_utils import to_categorical

one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
```

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

# vectorized training/test data
x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)

def to_one_hot(labels, dimension=46):
    results = np.zeros((len(labels), dimension))
    for i, label in enumerate(labels):
        results[i, label] = 1.
    return results

# vectorized training/test labels
one_hot_train_labels = to_one_hot(train_labels)
one_hot_test_labels = to_one_hot(test_labels)
```

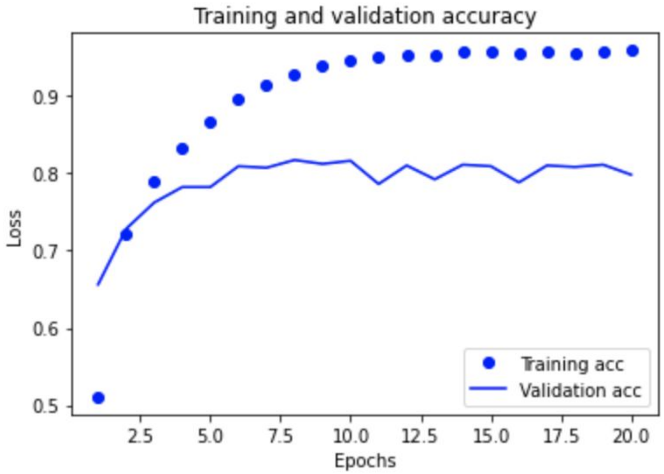
Modeling

- End the network with a Dense layer of size 46 using a softmax activation
- The network will produce a 46-dimensional output vector where output[i] is the probability that the sample belongs to class i. The 46 scores will sum to 1.
- The best loss function is **categorical_crossentropy**. It measures the distance between the probability distribution output by our network, and the true distribution of the labels

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))

model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

x_val = x_train[:1000]
partial_x_train = x_train[1000:]

y_val = one_hot_train_labels[:1000]
partial_y_train = one_hot_train_labels[1000:]

history = model.fit(partial_x_train, partial_y_train, epochs=20,
                    batch_size=512,   validation_data=(x_val, y_val))
```

Evaluation

- two intermediate layers with 16 hidden units each, and a third layer which will output the scalar prediction regarding the sentiment of the current review




```
plt.clf()   # clear figure

acc = history.history['acc']
val_acc = history.history['val_acc']

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()

print(val_acc[-1])  # 0.7979999966621399
results = model.evaluate(x_test, one_hot_test_labels)
print(results)  # [1.2319941453814824, 0.7822796081923421]
```

Conclusion

- If you are trying to classify data points between N classes, your network should e**nd with a Dense layer of size N**
- In a single-label, multi-class classification problem, your network should end with **a softmax activation**, so that it will output a probability distribution over the N output classes.
- **Categorical crossentropy** is almost always the loss function you should use for such problems. It minimizes the distance between the probability distributions output by the network, and the true distribution of the targets
- There are two ways to handle labels in multi-class classification: Encoding the labels via "categorical encoding" (also known as "one-hot encoding") and using categorical_crossentropy as your loss function. Encoding the labels as integers and using the sparse_categorical_crossentropy loss function
- If you need to classify data into a large number of categories, then you should **avoid creating information bottlenecks in your network by having intermediate layers that are too small**

Further analysis

- User Tf-idf
- Try to use 1 or 3 hidden layers and see how it affects validation and test accuracy
- Try to use layers with more hidden units or less hidden units: 32 units, 64 units...
- Try to use the mse loss function instead of binary_crossentropy
- Try to use the tanh activation (an activation that was popular in the early days of neural networks) instead of relu
- Vary the epoch