Assignment 2 - INF368

PHH - 21.09.2021

TASK:

The first task of the assignment is to write a program that uses the naïve bayes approach for sentiment classification.
Sentiment classification is the task of determining whether the sentiment of an input phrase is positive or negative (Although this can of course be expanded to include more classes). The value of such a system is best seen when inspecting the different use cases. It might for instance be used to inspect whether a movie is well received or not. Similarly it can be used to find the public opinion on different products and services, even political figures. It can even be used for prediction. A naïve bayes classifier is an example of a generative classifier. A Generative classifier explicitly models the actual distribution of each class. It will in short learn the joint probability distribution p(x,y) and predict the conditional probability with the help of Bayes Theorem[2].

The task was quite straightforward; Create a simple program that, given the training set on the right, outputs the predicted probability of a phrase D belonging to any of the classes. *(D = fast, couple, shoot, fly)*. We should also include add1-smoothing in our implementation.

1. fun, couple, love, love **comedy**
2. fast, furious, shoot **action**
3. couple, fly, fast, fun, fun **comedy**
4. furious, shoot, shoot, fun **action**
5. fly, fast, shoot, love **action**

The second task is to implement logistic regression. Logistic regression is a type of discriminative model which means that it models the decision boundary between the classes. This can again be used for sentiment classification, although it is more of a weighted sum of occurences than a pure probability as seen in the generative models. Logistic regression consists of a single layer of weights which the model must learn during a training step. This training step is done with gradient descent. A method where we iteratively update the weights in the direction that minimizes the loss. After a few epochs the model will have learned where to place the optimal boundary.

Design choices:

I took a few liberties when developing my programs to make use and development a bit easier.
The first thing I want to specify is that I assume the input (for both training and prediction) strings contain no special characters. This solves the problem of “love**,**” != “love”. Which essentially means I look away from the commas in the training data. Additionally I made the

tool non-case sensitive. Since the task contained such a simple dataset I did not see the value in using 50% of the development time to handle all the different edge cases.

I chose to implement my sentiment analysis tool as an object to make it easier to import into other projects. It has two main functions, *fit* and *predict*. Fit takes the training data as the only parameter which it requires is properly formatted. Each training example must be a tuple/list with two elements, the first element is the sentence (with special characters removed), the second is the ground truth label of the sentence.
Predict uses the information gathered during training/fitting to compute the prediction for each of the classes in the training data. During the fitting procedure the algorithm will gather all necessary knowledge about the training data to be able to make predictions. This includes the vocabulary, the number of training examples, which class each sentence corresponds with, and finally how many words are in each class.

P^{hat} is the probability of a given class.

$$\hat{P}\left(c_j\right) = \frac{N_{c_j}}{N_{total}}$$

$p(w_i | c)$ is the probability of a given word being in a certain class. This formula is used on all words in the sentence we want to classify. The +1 in the numerator comes from add-1 smoothing. I decided to make this an optional hyperparameter for added flexibility.

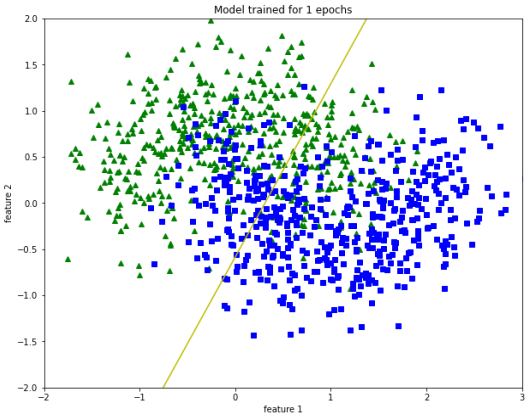
$$p\left(w_i \middle| c\right) = \frac{count\left(w_i, c\right) + 1}{\left(\sum_{w \in V} count(w, c)\right) + |V|}$$

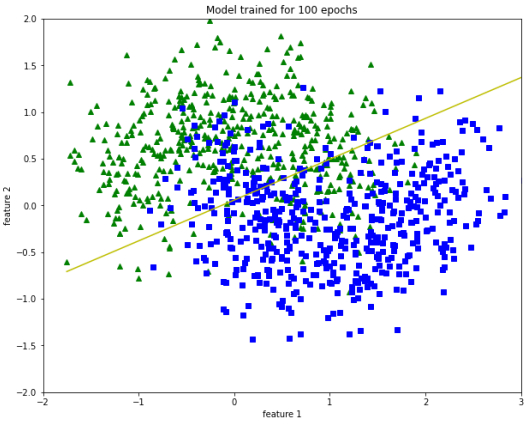
My **logistic regression** implementation is also an object for the same reason. The object has two main functions, *training_loop* and *predict*. The fitting procedure “training_loop” will fit the model optimally to the training data. It is a quite simple SGD implementation. First instantiate the object, then train it on your training data, now you can make predictions. I added an option to visualize the decision boundary with some heavy inspiration from [1].

Evaluation:

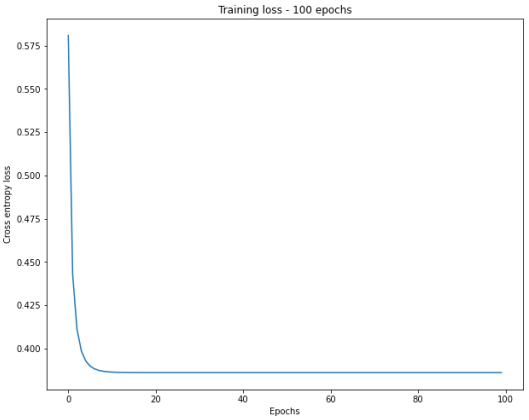
With the sentiment analysis tool there is not much to say really, it seems to work alright. Hopefully for you as well. I tested it on the exercise where we computed the naïve bayes by hand and got the same results. And when comparing with other colleagues it seemed we got the same prediction for D.

It is more fun to inspect the results of the logistic regression as we can plot the results :))





Here we can see that the model does not fit the data optimally after the first epoch. But after 100 epochs it does quite well!
I was curious to see how many epochs were necessary to get good results. For this specific dataset it seems to be sufficient with around 10 epochs.



Installation:

The tools are found in different files, as well as in the notebook. Everything is made in Python3, and is untested in Python2. The sentiment analysis tool is built in pure python and has no dependencies. The logistic regression tool uses both numpy and matplotlib. Although it can be modified to only use numpy by removing the plotting function.

- Install Python3
    - https://www.python.org/downloads/
- Install numpy
    - (Windows) pip install numpy
    - (Linux) sudo pip install numpy
    - Test installation: run python and import numpy
- Install Matplotlib
    - (Windows) pip install Matplotlib
    - (Linux) sudo pip install Matplotlib
    - Test installation: run python and import Matplotlib

To use the tools in a new project simply do the following:

- Import the tool
- Instantiate an object
- Fit the object to training data
- Make predictions

Logistic regression:

```
from sklearn.datasets import make_moons
import numpy as np
from LogisticRegression import LogReg

X, y = make_moons(n_samples=1000, noise=0.4)   # Example dataset
regressor = LogReg(2)
regressor.training_loop(X,y, epochs=1)
regressor.plot_decision_boundary(X, y,"Model trained for 1 epochs")
regressor.predict(np.array([0,2]))
```

It is also possible to change the learning rate and the number of epochs with optional input parameters *lr* and *epochs*.

Sentiment Analysis:

```
from SentimentAnalysis import SentimentAnalysis
naive_bayes = SentimentAnalysis()
data = [("fun couple love love", "comedy"),
        ("fast furious shoot", "action"),
        ("couple fly fast fun fun", "comedy"),
        ("furious shoot shoot fun", "action"),
        ("fly fast shoot love", "action")]
naive_bayes.fit(data)
print(naive_bayes.predict("fast couple shoot fly"))
```

It is possible to change the k for the add-k smoothing. It defaults to 1. But if required you can input a parameter *k* with a different k.

Sources:

1. https://towardsdatascience.com/logistic-regression-from-scratch-in-python-ec66603592e2
2. https://medium.com/@mlengineer/generative-and-discriminative-models-af5637a66a3