Homework 1:
Spam Classification and the Perceptron

Dr. Benjamin Roth
Computerlinguistische Anwendungen

Due: Wednesday May 8, 2019, 14:00

In this homework you will review important basics about data representation, classification and the perceptron. The result will be a spam detector.

The main file for this homework is `hw01_perceptron/utils.py`: It contains functionality to store and process datasets. Changes to the perceptron classifier need to be made in the file `hw01_perceptron/perceptron_classifier.py`.

This homework will be graded using unit tests. You can run the tests from the `src` directory with:

```
python3 -m unittest -v hw01_perceptron/test_perceptron.py
```

The homework must be submitted via git. Do not send it by email. To submit, you can for example use this sequence of git commands:

```
git commit -a -m "Homework 1 submission"
git pull
git push
```

Exercise 1: Setting up the Git project

In order to have access to the Git project with the exercise code, and to be able to submit your solution, you need to do the following steps (ask the tutors if any of the steps is unclear to you):

1. Make sure you have a Gitlab account for `gitlab2.cip.ifi.lmu.de`
2. Form teams of 2 or 3 students (4 students are not allowed).
3. Use the web form (which you can find on `cla2019.github.io`) to provide the following information for your group:
   - A team name you choose (please use only letters, numbers, underscore)
   - Gitlab id, name, and matriculation number of each team member

4. We will then create a project in Gitlab for you, that will contain the code to the exercises. Submit your solution by pushing to this project.

5. Please do not create separate files or folders to submit your solution. Instead, change the files we provided.

6. Exercises will be mainly graded by unit tests. Important: do NOT change the tests themselves, implement the missing functionality instead. Changing the tests will result in your exercise sheet scored with 0 points.

Exercise 2: Dot product [2 Point]

Complete the function `dot(dictA, dictB)` in `utils.py`, which computes the dot product (vector product) between two dictionaries with feature counts.

Exercise 3: Creating a data instance [4 Points]

The `DataInstance` class is used to store a data instance (e.g. a training example). It holds for each instance the label of the instance and a dictionary that maps strings (the features) to numbers (the feature value). Complete the class method.

`DataInstance.from_list_of_feature_occurrences(...)`, that takes a list of features (possibly unordered and/or with repetitions), and creates an instance where the feature values are the counts of the features.

Exercise 4: Most frequent features [4 Points]

Often only the most frequent features are used in classification, since infrequent features may lead to overfitting. Complete the function `get_topn_features(...)` that outputs a set of the `n` most frequent features for a data set (`DataSet` is used for grouping several `DataInstance`s). The *frequency* of a feature is measured by the number of instances it occurs in.

Exercise 5: Filtering features [4 Points]

Provide the functionality to restrict the features used in a dataset and complete the method `set_feature_set(...)` so that the data set and all instances only contain the specified features.

Exercise 6: Most frequent sense baseline [2 Points]

Complete the function `DataSet.most_frequent_sense_accuracy(...)` that computes the accuracy for always predicting the most frequent label in a data set.

Exercise 7: Perceptron Update [2 Points]

During training the weights of the perceptron classifier are iteratively adjusted by performing the so called perceptron update.

- If the classifier's prediction for a training instance is already correct, do nothing.
- Otherwise increase/reduce the weights of your classifier so that they better fit the training data. Keep in mind that negative weights are possible.

Complete the method `PerceptronClassifier.update(self, instance)`.
In this method you only have to replace two lines:

- Replace `error = 0` with the correct calculation of the error.
- In the for loop replace the `pass` statement with the correct update of feature weights.

Exercise 8: F-Measure [4 Points]

Complete the method `PerceptronClassifier.prediction_f_measure(...)` that computes the f-measure for a given classifier, data set and label of interest.

Exercise 9: Using the classifier

Once you have implemented all missing functionality, you can train and evaluate the classifier on an actual data set of emails and spam. On the course homepage, you can find a dataset of corporate emails[1], containing several folders of spam or normal (ham) emails. Download and unpack it into the `src/data/` folder of your project. Have a look at the file `spam_classification` to see how the data is processed, and how the model is trained. You can call training with:

```
python3 -m hw01_perceptron.spam_classification -p data/enron/enron_utf8/enron1/ham/
-n
data/enron/enron_utf8/enron1/spam/ -pp data/enron/enron_utf8/enron2/ham/ -nn
data/enron/enron_utf8/enron2/spam/ -ppp data/enron/enron_utf8/enron3/ham/ -nnn
data/enron/enron_utf8/enron3/spam
```

[1] See https://en.wikipedia.org/wiki/Enron_Corpus for the history of this dataset