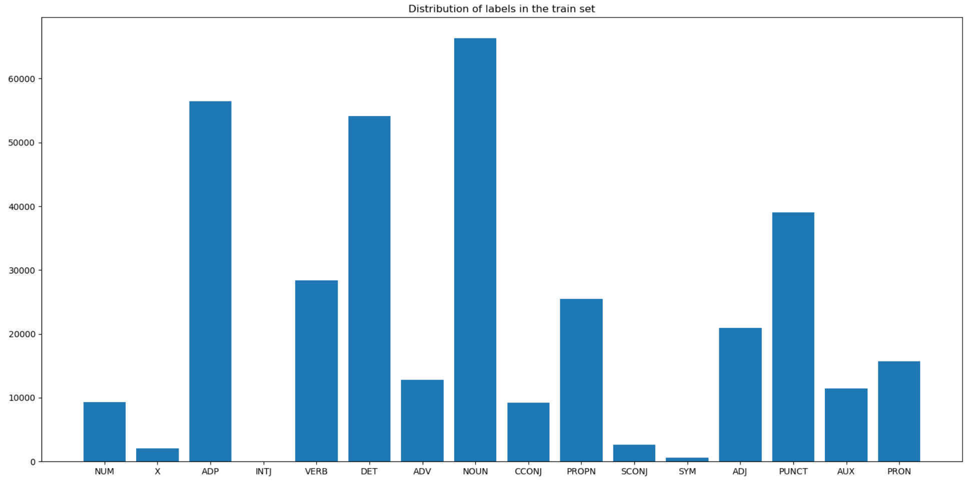
PoS tagging with a perceptron

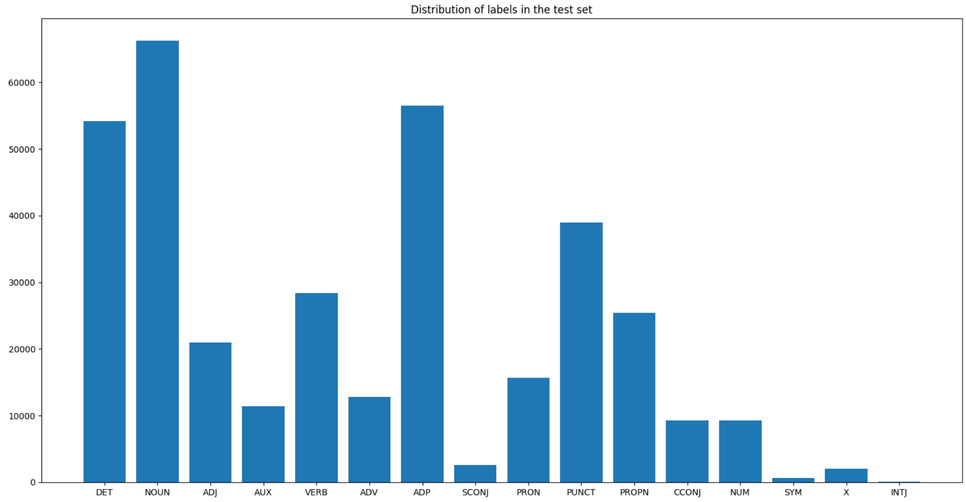
Questions

1. **Corpus reading**

The training set has 14449 examples.



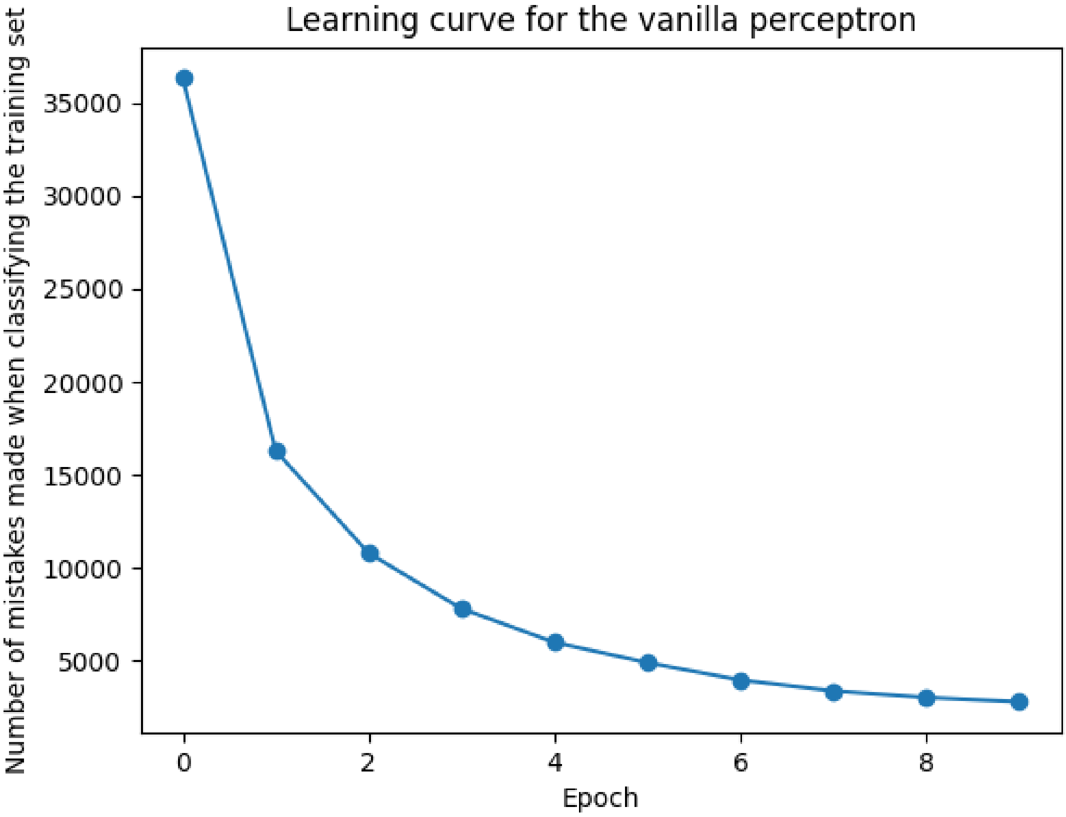
The test set has 416 examples.

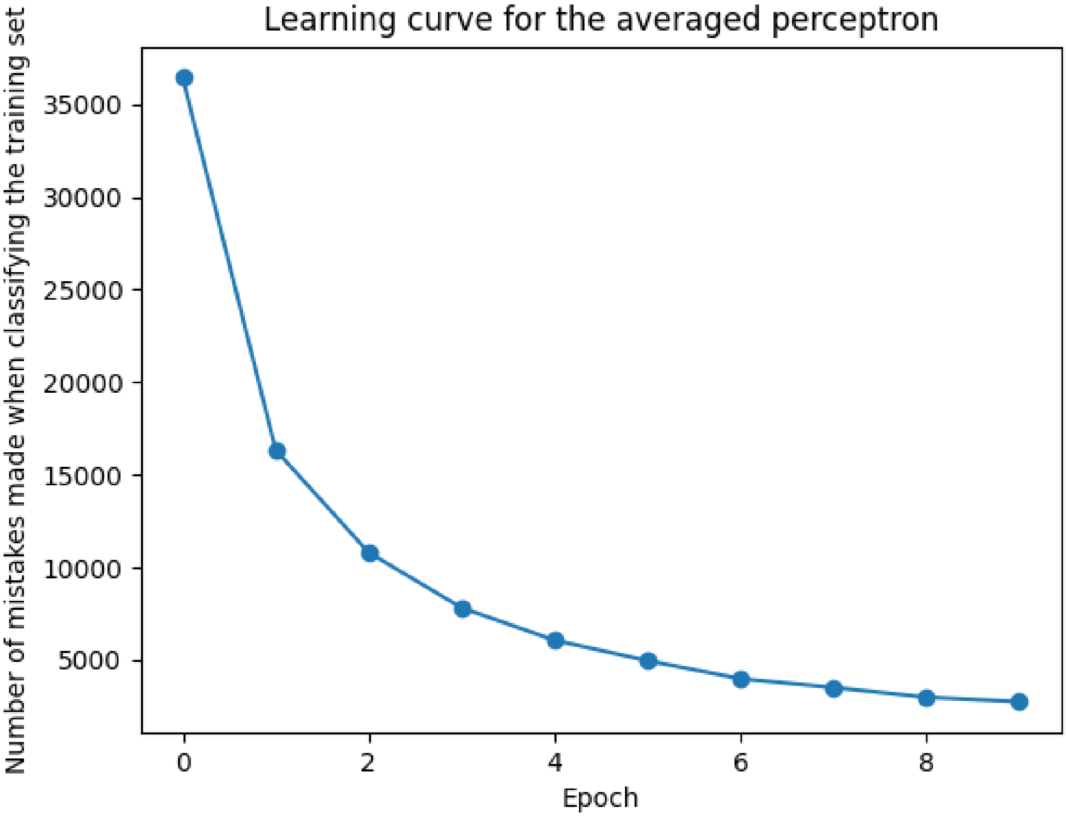


2. **Feature extraction**

The dimension of the feature vector is 8, but feature vectors only have an average of 5,9 non-zero values.

3. **Averaged perceptron implementation**





4. **Evaluation**

The accuracy of the vanilla perceptron is of 94.82%.

The accuracy of the averaged perceptron is of 95.82%.

Code for the vanilla perceptron

```python
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
import random
from collections import defaultdict as defaultdict

# note to self: factoriser le code

def parse_sentence(lines):
    words = list()
    labels = list()
    # the expected word number corresponds
    # to the first number in word lines on the .conllu file
    word_number = 1
    for line in lines:
        columns = line.split("\t")
        if columns[0] == str(word_number):
            words.append(columns[1])
            labels.append(columns[3])
            word_number += 1
    return words, labels

def corpus_reading(file_name):
    list_examples = list()
    file = open(file_name, "r")
    list_sentences = file.read().split("\n\n")
    for sentence in list_sentences:
        lines = sentence.split("\n")
        words, labels = parse_sentence(lines)
        if words:    # if sentence is not empty
            list_examples.append((words, labels))
    file.close()
    return list_examples

training_set = corpus_reading("/media/lina/RED/TP2ML/fr_gsd-ud-train.conllu")
#training_set = corpus_reading("D:\TP2ML\fr_gsd-ud-train.conllu")
print("the training set has " + str(len(training_set)) + " examples.")


label_dictionary = defaultdict(int)
for example in training_set:
    for label in example[1]:
        label_dictionary[label] += 1

'''
plt.title("Distribution of labels in the train set")

xpoints = list(label_dictionary.keys())
ypoints = list(label_dictionary.values())
plt.bar(xpoints,ypoints)
plt.show()
```

```
'''

test_set = corpus_reading("/media/lina/RED/TP2ML/fr_gsd-ud-test.conllu")
#test_set = corpus_reading("D:\TP2ML\fr_gsd-ud-test.conllu")
print("the test set has " + str(len(test_set)) + " examples.\n")


label_dictionary = defaultdict(int)
for example in training_set:
    for label in example[1]:
        label_dictionary[label] += 1

'''
plt.title("Distribution of labels in the test set")

xpoints = list(label_dictionary.keys())
ypoints = list(label_dictionary.values())
plt.bar(xpoints,ypoints)
plt.show()
'''

def feature_extraction(list_observation):
    list_representation = list()
    for sentence, label in list_observation:
        for i in range(len(sentence)):
            feature_vector = list()
            feature_vector.append("curr_word_" + sentence[i])
            if i > 0:
                feature_vector.append("prev_word_" + sentence[i-1])
            if i > 1:
                feature_vector.append("prev_prev_word_" + sentence[i-2])
            if len(sentence) - i > 1:
                feature_vector.append("next_word_" + sentence[i+1])
            if len(sentence) - i > 2:
                feature_vector.append("next_next_word_" + sentence[i+2])
            feature_vector.append("biais")
            if sentence[i][0].isupper():
                feature_vector.append("starts_with_upper")
            if True in [char.isdigit() for char in sentence[i]]:
                feature_vector.append("contains_number")
            list_representation.append((feature_vector, label[i]))
    return list_representation

training_observations = feature_extraction(training_set)


def dot_sparse(observation_vector, parameters_vector):
    return sum(parameters_vector.get(key, 0) for key in observation_vector)


class Perceptron:

    def __init__(self, labelset):
        ''' parametres est une liste contenant, pour chaque label,
        un tuple de la forme (label, parameter_vector_of_the_label) '''
        self.parameters = defaultdict(lambda: defaultdict(int))
        for label in labelset:
            self.parameters[label]
    def predict(self, observation):
        return max((dot_sparse(observation, self.parameters[label]), label) for
label in self.parameters.keys())[1]

    def fit(self, training_observations, n_epochs, listener):
        for epoch in range(n_epochs):
            random.shuffle(training_observations)
            nb_mistakes = 0
            for observation, gold_label in training_observations:
                predicted_label = self.predict(observation)
                if predicted_label != gold_label:
                    nb_mistakes += 1
                    for feature in observation:
                        self.parameters[gold_label][feature] += 1
                        self.parameters[predicted_label][feature] -= 1
            listener(epoch, nb_mistakes)

    def score(self, test_observations):
        return sum((self.predict(observation) == gold_label) for observation,
gold_label in test_observations)*100/len(test_observations)


c = Perceptron(label_dictionary.keys())

def print_mistakes_when_training(epoch,nb_mistakes):
    print("epoch " + str(epoch + 1) + ": " + str(nb_mistakes) + " mistakes")

xpoints = list()
ypoints = list()
def plot_mistakes_when_training(epoch,nb_mistakes):
    xpoints.append(epoch)
    ypoints.append(nb_mistakes)

c.fit(training_observations, 10, plot_mistakes_when_training)

print("\naccuracy: " + str(c.score(feature_extraction(test_set))) + "%\n")

plt.title("Learning curve for the vanilla perceptron")
plt.xlabel("Epoch")
plt.ylabel("Number of mistakes made when classifying the training set")

plt.plot(xpoints, ypoints, marker = 'o')
plt.show()

'''
# testing on a small hand-made example
test_observations = feature_extraction([(["Le", "chat", "est", "triste", "."],
["DET", "NOUN", "VERB", "ADJ", "PUNCT"])])

for o,l in test_observations:
    print("predictated label: " + c.predict(o) + "\tgold label: " + l)
'''
```

Code for the averaged perceptron

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
import random
from collections import defaultdict as defaultdict

# try to make functions less than 10 lines long (factoriser le code) + remove
cpmments before I send it + plotting the evolution of the number of mistakes for
each epoch

def parse_sentence(lines):
    words = list()
    labels = list()
    # the expected word number corresponds
    # to the first number in word lines on the .conllu file
    word_number = 1
    for line in lines:
        columns = line.split("\t")
        if columns[0] == str(word_number):
            words.append(columns[1])
            labels.append(columns[3])
            word_number += 1
    return words, labels


def corpus_reading(file_name):
    list_examples = list()
    file = open(file_name, "r")
    list_sentences = file.read().split("\n\n")
    for sentence in list_sentences:
        lines = sentence.split("\n")
        words, labels = parse_sentence(lines)
        if words:    # if sentence is not empty
            list_examples.append((words, labels))
    file.close()
    return list_examples

training_set = corpus_reading("/media/lina/RED/TP2ML/fr_gsd-ud-train.conllu")
print("the training set has " + str(len(training_set)) + " examples.")


label_dictionary = defaultdict(int)
for example in training_set:
    for label in example[1]:
        label_dictionary[label] += 1

'''
plt.title("Distribution of labels in the train set")

print(list(label_dictionary.keys()))
print(list(label_dictionary.values()))
xpoints = list(label_dictionary.keys())
ypoints = list(label_dictionary.values())
plt.bar(xpoints,ypoints)
plt.show()
'''
```

```
test_set = corpus_reading("/media/lina/RED/TP2ML/fr_gsd-ud-test.conllu")
print("the test set has " + str(len(test_set)) + " examples.\n")

'''
label_dictionary = defaultdict(int)
for example in training_set:
    for label in example[1]:
        label_dictionary[label] += 1

plt.title("Distribution of labels in the test set")

print(list(label_dictionary.keys()))
print(list(label_dictionary.values()))
xpoints = list(label_dictionary.keys())
ypoints = list(label_dictionary.values())
plt.bar(xpoints,ypoints)
plt.show()
'''

def feature_extraction(list_observation):
    list_representation = list()
    for sentence, label in list_observation:
        for i in range(len(sentence)):
            feature_vector = list()
            feature_vector.append("curr_word_" + sentence[i])
            if i > 0:
                feature_vector.append("prev_word_" + sentence[i-1])
            if i > 1:
                feature_vector.append("prev_prev_word_" + sentence[i-2])
            if len(sentence) - i > 1:
                feature_vector.append("next_word_" + sentence[i+1])
            if len(sentence) - i > 2:
                feature_vector.append("next_next_word_" + sentence[i+2])
            feature_vector.append("biais")
            if sentence[i][0].isupper():
                feature_vector.append("starts_with_upper")
            if True in [char.isdigit() for char in sentence[i]]:
                feature_vector.append("contains_number")
            list_representation.append((feature_vector, label[i]))
    return list_representation

training_observations = feature_extraction(training_set)

def dot_sparse(observation_vector, parameters_vector):
    return sum(parameters_vector.get(key, 0) for key in observation_vector)


class AveragedPerceptron:

    def __init__(self, labelset):
        ''' parametres est une liste contenant, pour chaque label,
        un tuple de la forme (label, parameter_vector_of_the_label) '''
        self.w_parameters = defaultdict(lambda: defaultdict(int))
        self.a_parameters = defaultdict(lambda: defaultdict(int))
        for label in labelset:
            self.w_parameters[label]
            self.a_parameters[label]
    def w_predict(self, observation):
        return max((dot_sparse(observation, self.w_parameters[label]), label) for
label in self.w_parameters.keys())[1]

    def fit(self, training_observations, n_epochs, listener):
        ''' date_last_update: dict(label, feature) -> number of examples we had
seen the last time we modified this value
        this makes the code more efficient because we don't need to increment a
value on the counter for each feature
        when the predicted label is correct, instead we only modify the counter
dictionnary when we make a mistake '''
        date_last_update = defaultdict(int)
        nb_seen_examples = 0
        for epoch in range(n_epochs):
            random.shuffle(training_observations)
            nb_mistakes = 0
            for observation, gold_label in training_observations:
                predicted_label = self.w_predict(observation)
                if predicted_label != gold_label:
                    nb_mistakes += 1
                    for feature in observation:
                        self.a_parameters[gold_label][feature] +=
self.w_parameters[gold_label][feature] * (nb_seen_examples -
date_last_update[gold_label, feature])
                        self.a_parameters[predicted_label][feature] +=
self.w_parameters[predicted_label][feature] * (nb_seen_examples -
date_last_update[predicted_label, feature])

                        self.w_parameters[gold_label][feature] += 1
                        self.w_parameters[predicted_label][feature] -= 1

                        date_last_update[gold_label, feature] = nb_seen_examples
                        date_last_update[predicted_label, feature] =
nb_seen_examples
                nb_seen_examples += 1
            listener(epoch, nb_mistakes)
        for label, feature in date_last_update:
            self.a_parameters[label][feature] += self.w_parameters[label]
[feature] * (nb_seen_examples - date_last_update[label, feature])

    def a_predict(self, observation):
        return max((dot_sparse(observation, self.a_parameters[label]), label) for
label in self.w_parameters.keys())[1]


    def score(self, test_observations):
        return sum((self.a_predict(observation) == gold_label) for observation,
gold_label in test_observations)*100/len(test_observations)

c = AveragedPerceptron(label_dictionary.keys())

def print_mistakes_when_training(epoch,nb_mistakes):
    print("epoch " + str(epoch + 1) + ": " + str(nb_mistakes) + " mistakes")

xpoints = list()
ypoints = list()
def plot_mistakes_when_training(epoch,nb_mistakes):
    xpoints.append(epoch)
    ypoints.append(nb_mistakes)

c.fit(training_observations, 10, plot_mistakes_when_training)


print("\naccuracy: " + str(c.score(feature_extraction(test_set))) + "%\n")

plt.title("Learning curve for the averaged perceptro")
plt.xlabel("Epoch")
plt.ylabel("Number of mistakes made when classifying the training set")

plt.plot(xpoints, ypoints, marker = 'o')
plt.show()

'''
# testing on a small hand-made example
test_observations = feature_extraction([(["Le", "chat", "est", "triste", "."],
["DET", "NOUN", "VERB", "ADJ", "PUNCT"])])

for o,l in test_observations:
    print("predicted label: " + c.a_predict(o) + "\tgold label: " + l)
'''
```