Linguistic Probes with l1-Regularization

Lina Conti M1 LI

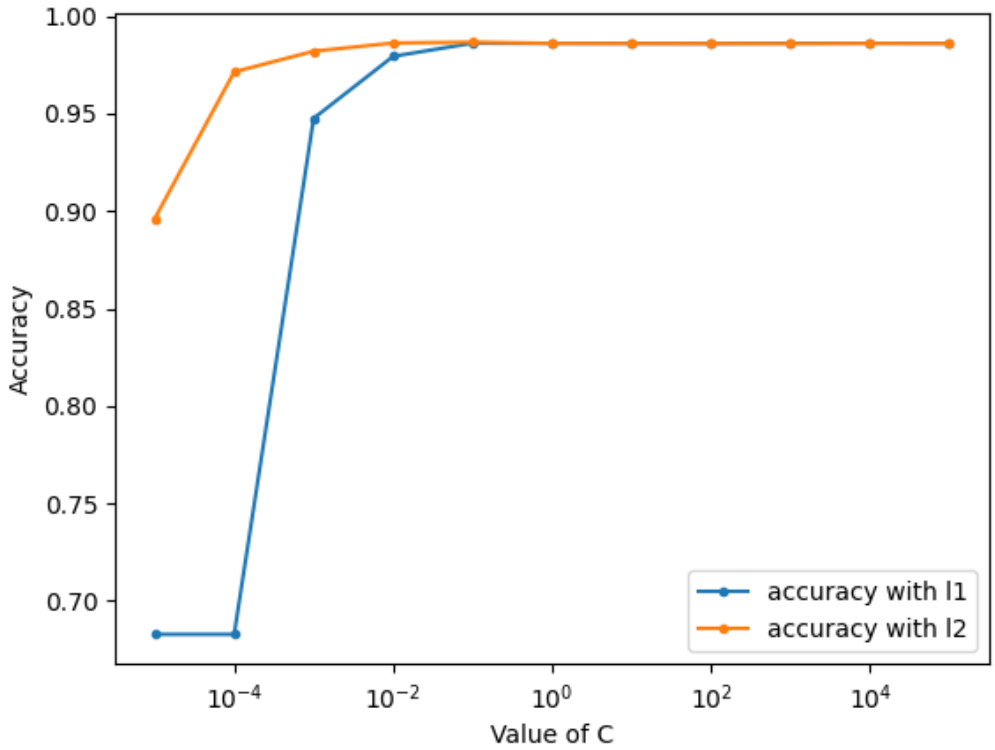
1. 1. Iel nous a écoutés.
   2. Iel nous a parlé.
   3. Ces fruits sont délicieux ; je les ai achetés.
   4. Les mots que J.-P. Sartre a écrits à S. de Beauvoir ont été publiés sous le titre *Lettres au Castor*.
2. Examples 3-4 are more interesting because in these sentences the direct object refers to another prior noun phrase ("les" refers to "ces fruits", "que" refers to "les mots") so in order to choose the correct agreement the neural network would have to correctly associate three elements: the verb, the direct object and the NP it refers to. Examples 1-2 are less interesting because here only the direct object is relevant.
3. We do not consider gender agreement because we want to focus on one simple linguistic property to see if it is encoded in the representation, not two.
4. The executables installed in this virtual environment are *activate , blpk, f2py3, pip3, pylupdate5, pyuic5, activate.csh, easy_install, f2py3.8, pip3.8, pyrcc5, ttx, activate.fish, easy_install-3.8, fonttools, pyftmerge, python, Activate.ps1, f2py, pip, pyftsubset* and *python3*. They are all contained in ./local/bin.
5. pip is the package installer for Python. It is installing the libraries halo, sklearn and bloscpack to the virtual environment (they will be stored in /local).
6. Without specifying any path, running python will run the system-wide python command. In this case, running import bloscpack will raise the Exception "ImportError" and show the message "No module named bloscpack". This is because the module was installed only to the virtual environment and system-wide python will therefore not be able to find it among its libraries. To avoid this problem, we must run ./local/bin/python instead of just python.
7. There are 64421 examples and 768 features in the corpus.
8.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,
random_state=42)
```

9.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.linear_model import LogisticRegression
from halo import Halo
import pandas as pd
import matplotlib.pyplot as plt
```

```
c_list = [10 ** i for i in range(-5, 6)]

accuracy_list_l1 = list()
accuracy_list_l2 = list()

for c in c_list:

    print("------------ C = " + str(c) + " ------------")

    with Halo(text="Training l1", spinner='dots') as spinner:
        clf_l1 = LogisticRegression(penalty = 'l1', solver = 'saga', C=c)
        clf_l1.fit(X_train, y_train)
    spinner.succeed("Training l1")

    print(clf_l1.score(X_test, y_test))
    accuracy_list_l1.append(clf_l1.score(X_test, y_test))

    with Halo(text="Training l2", spinner='dots') as spinner:
        clf_l2 = LogisticRegression(penalty = 'l2', solver = 'saga', C=c)
        clf_l2.fit(X_train, y_train)
    spinner.succeed("Training l2")

    print(clf_l2.score(X_test, y_test))
    accuracy_list_l2.append(clf_l2.score(X_test, y_test))

df1 = pd.DataFrame({"Value of C": c_list, "accuracy with l1":
accuracy_list_l1, "accuracy with l2": accuracy_list_l2})
print(df1)

df1.plot(x = "Value of C", y = {"accuracy with l1", "accuracy with l2"},
ylabel = "Accuracy", style = '.-')
plt.xscale('log')
plt.show()
```



The accuracy of the classifier increases along with the value of C. This is because by increasing C, we allow it to use a more complex classifying function that will be able to fit the data better. However, the accuracy seems to stagnate from C = 0,1 onwards. If we look closely at the data, the accuracy even decreases slightly after that. This might be due to overfitting: the function matches the training data so closely that it will not be able to generalize on new data it has never seen (the test set). It could for example just be memorizing the training data. Therefore, the best choice would be to use the classifier found for C = 0,1.

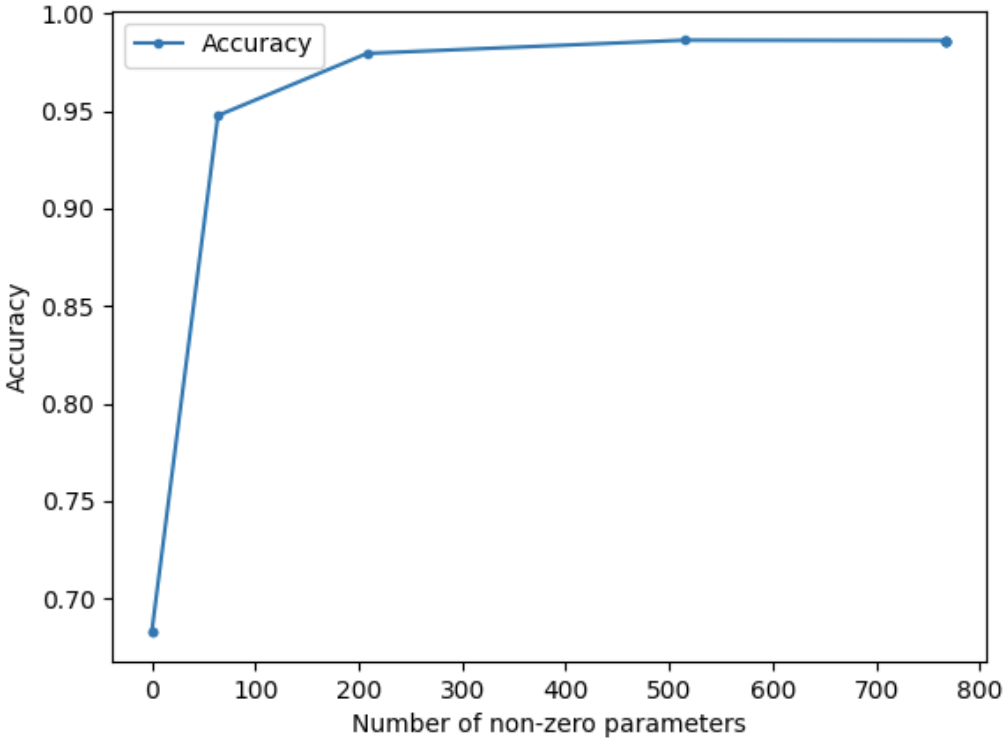
What we have said before holds both for l1 regularization and l2 regularization. However, the accuracies obtained with l2 are slightly better than those obtained with l1, and even a lot better for small values of C. This is because l2 is easier to optimize than l1. Still, we will be using l1 regularization for the rest of this work, because we want to have a sparse solution with as few parameters as possible, so we can try to understand how the classifier works and into which features the information we are interested in is encoded.

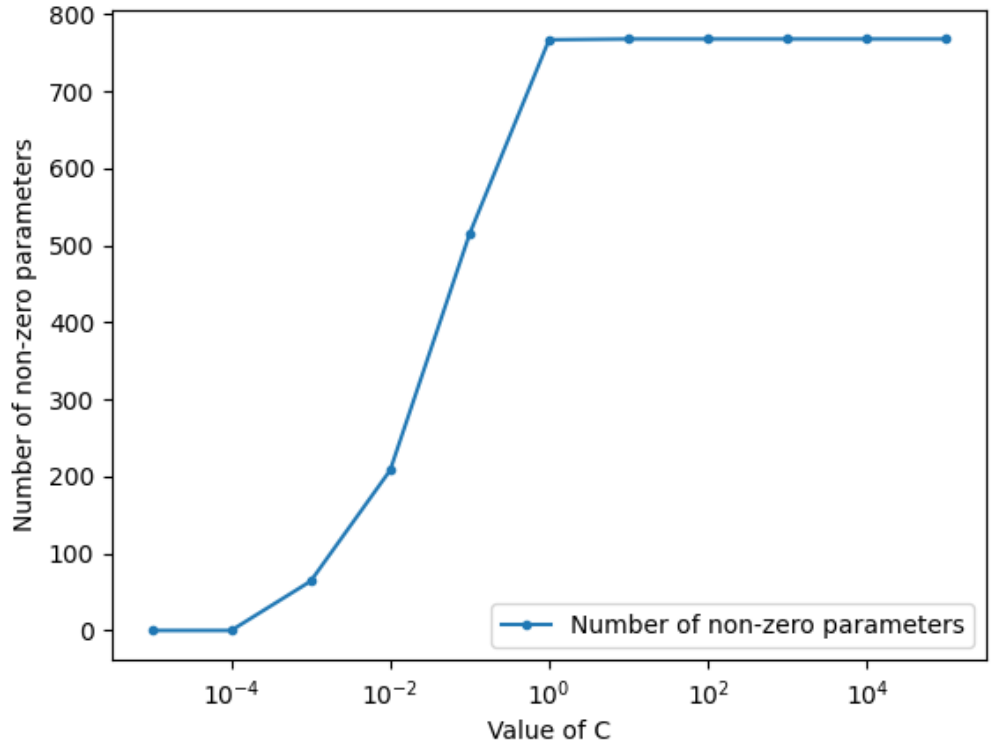
10. C cannot be found in the same way other parameters are found because it is a hyper-parameter, it controls the complexity of the function. So changing C will change the number of features. When C is changed, all previous work done on fitting the function and finding the best weight vector will be lost, so C cannot be modified while training the classifier.

    To choose the optimal value of C, we need to train a classifier for each possible value and evaluate its performance on the test set, then we choose the value of C with the smallest test error, that is with the best accuracy.

11.
```
import bloscpack
X = bloscpack.unpack_ndarray_from_file("correct_context_que_X.blp")
y = bloscpack.unpack_ndarray_from_file("correct_context_que_Y.blp")

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8,
random_state=42)
```

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

from sklearn.linear_model import LogisticRegression
from halo import Halo
import pandas as pd
import matplotlib.pyplot as plt

c_list = [10 ** i for i in range(-5, 6)]

accuracy_list_l1 = list()
non_zero_parameters_list = list()

for c in c_list:

    print("------------ C = " + str(c) + " ------------")

    with Halo(text="Training l1", spinner='dots') as spinner:
        clf_l1 = LogisticRegression(penalty = 'l1', solver = 'saga', C=c)
        clf_l1.fit(X_train, y_train)
    spinner.succeed("Training l1")

    print(clf_l1.score(X_test, y_test))
    accuracy_list_l1.append(clf_l1.score(X_test, y_test))
    nb_non_zero_parameters = sum(x != 0 for x in clf_l1.coef_[0])
    non_zero_parameters_list.append(nb_non_zero_parameters)

df2 = pd.DataFrame({"Number of non-zero parameters":
non_zero_parameters_list, "Accuracy": accuracy_list_l1})
print(df2)

df2.plot(x = "Number of non-zero parameters", ylabel = "Accuracy", style =
'.-')
plt.show()

df3 = pd.DataFrame({"Value of C": c_list, "Number of non-zero parameters":
non_zero_parameters_list})
print(df3)

df3.plot(x = "Value of C", ylabel = "Number of non-zero parameters", style =
'.-')
plt.xscale('log')
plt.show()
```





The accuracy increases with the number of non-zero parameters and the number of non-zero parameters increases with the value of C. We can conclude that information about the number of the verb is encoded in the feature representation of the "que" token, since the non-zero parameters of this representation make it possible to predict the number of the verb with a good accuracy, higher than 97%.

From 515 non-zero parameters on, the accuracies get much higher than before. However, the best accuracy is achieved with only 515 non-zero parameters and decreases very slightly afterwards. We can therefore emit the hypothesis that only about 500 features are useful for predicting the agreement of the past participle with the direct object and the rest are not

relevant for determining this information, but this hypothesis needs to be further tested.

12.

13.

14.

```
import bloscpack
X = bloscpack.unpack_ndarray_from_file("correct_context_que_X.blp")
y = bloscpack.unpack_ndarray_from_file("correct_context_que_Y.blp")

from sklearn.model_selection import train_test_split

X_train_original, X_test_original, y_train, y_test = train_test_split(X, y,
train_size=0.8, random_state=42)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train_original)

X_train_original = scaler.transform(X_train_original)
X_test_original = scaler.transform(X_test_original)

from sklearn.linear_model import LogisticRegression
from halo import Halo
import pandas as pd
import matplotlib.pyplot as plt
import numpy

c_list = [10 ** i for i in range(-5, 6)]

accuracies_by_c = list()
features_by_c = list()

for c in c_list:

    print("------------ C = " + str(c) + " ------------")

    accuracy_list= list()
    parameters_list = list()
    nb_zero_parameters = 1  # juste pour l'initialiser à quelque chose
    X_train = X_train_original
    X_test = X_test_original

    while True:

        with Halo(text="Training", spinner='dots') as spinner:
            clf_l1 = LogisticRegression(penalty = 'l1', solver = 'saga', C=c)
            clf_l1.fit(X_train, y_train)
        spinner.succeed("Training")

        print("Number of features: " + str(X_train.shape[1]))
        parameters_list.append(X_train.shape[1])
        acc = clf_l1.score(X_test, y_test)
        accuracy_list.append(acc)

        mask = clf_l1.coef_[0] == 0

        if True not in mask:
```

```
            print("Number of zero features: 0")
            print("Accuracy: " + str(acc))
            print("There are no zero parameters: break")
            break

        X_train = numpy.array(list(f[mask] for f in X_train))
        X_test = numpy.array(list(f[mask] for f in X_test))

        print("Number of zero features: " + str(X_train.shape[1]))
        print("Accuracy: " + str(acc))
        print()

        if nb_zero_parameters == X_train.shape[1]:
            print("The number of zero parameters has not changed: break")
            break
        nb_zero_parameters = X_train.shape[1]

    accuracies_by_c.append(accuracy_list)
    features_by_c.append(parameters_list)

print(accuracies_by_c)
print(features_by_c)
```

```
---------- C = 1e-05 ------------
Number of features: 768
Number of zero features: 768
Accuracy: 0.682731858750485

Number of features: 768
Number of zero features: 768
Accuracy: 0.682731858750485

The number of zero parameters has not changed: break

------------ C = 0.0001 ------------
Number of features: 768
Number of zero features: 768
Accuracy: 0.682731858750485

Number of features: 768
Number of zero features: 768
Accuracy: 0.682731858750485

The number of zero parameters has not changed: break

------------ C = 0.001 ------------
Number of features: 768
Number of zero features: 704
Accuracy: 0.9476135040745053

Number of features: 704
Number of zero features: 613
Accuracy: 0.9203725261932479

Number of features: 613
Number of zero features: 505
Accuracy: 0.8873884361660846
```

```
Number of features: 505
Number of zero features: 417
Accuracy: 0.8257663950329841

Number of features: 417
Number of zero features: 344
Accuracy: 0.7566938300349243

Number of features: 344
Number of zero features: 294
Accuracy: 0.6973224679860303

Number of features: 294
Number of zero features: 259
Accuracy: 0.6833527357392316

Number of features: 259
Number of zero features: 235
Accuracy: 0.682731858750485

Number of features: 235
Number of zero features: 222
Accuracy: 0.682731858750485

Number of features: 222
Number of zero features: 213
Accuracy: 0.682731858750485

Number of features: 213
Number of zero features: 201
Accuracy: 0.682731858750485

Number of features: 201
Number of zero features: 195
Accuracy: 0.682731858750485

Number of features: 195
Number of zero features: 189
Accuracy: 0.682731858750485

Number of features: 189
Number of zero features: 184
Accuracy: 0.682731858750485

Number of features: 184
Number of zero features: 181
Accuracy: 0.682731858750485

Number of features: 181
Number of zero features: 179
Accuracy: 0.682731858750485

Number of features: 179
Number of zero features: 176
Accuracy: 0.682731858750485

Number of features: 176
```

```
Number of zero features: 175
Accuracy: 0.682731858750485

Number of features: 175
Number of zero features: 175
Accuracy: 0.682731858750485

The number of zero parameters has not changed: break

------------ C = 0.01 ------------
Number of features: 768
Number of zero features: 561
Accuracy: 0.9794334497477687

Number of features: 561
Number of zero features: 312
Accuracy: 0.969499417927823

Number of features: 312
Number of zero features: 100
Accuracy: 0.9034536282499029

Number of features: 100
Number of zero features: 17
Accuracy: 0.7449747768723322

Number of features: 17
Number of zero features: 2
Accuracy: 0.6788513775708188

Number of features: 2
Number of zero features: 0
Accuracy: 0.682731858750485
There are no zero parameters: break

------------ C = 0.1 ------------
Number of features: 768
Number of zero features: 276
Accuracy: 0.9862630966239814

Number of features: 276
Number of zero features: 19
Accuracy: 0.9473030655801319

Number of features: 19
Number of zero features: 0
Accuracy: 0.7057819169577028
There are no zero parameters: break

------------ C = 1 ------------
Number of features: 768
Number of zero features: 0
Accuracy: 0.9859526581296081
There are no zero parameters: break

------------ C = 10 ------------
Number of features: 768
Number of zero features: 0
```

```
Accuracy: 0.9861078773767947
There are no zero parameters: break

------------ C = 100 ------------
Number of features: 768
Number of zero features: 0
Accuracy: 0.9861078773767947
There are no zero parameters: break

------------ C = 1000 ------------
Number of features: 768
Number of zero features: 0
Accuracy: 0.986185487000388
There are no zero parameters: break

------------ C = 10000 ------------
Number of features: 768
Number of zero features: 0
Accuracy: 0.9859526581296081
There are no zero parameters: break

------------ C = 100000 ------------
Number of features: 768
Number of zero features: 0
Accuracy: 0.9860302677532014
There are no zero parameters: break
```

If C is inferior to 0,001 all of the features are always set to zero. The accuracy is always 0,682731858750485 which we can assume is the accuracy obtained by always giving the most probable label (sing).

For C = 0,001 at first the number of zero features is 704 and the accuracy is 0,9476135040745053. If we train a classifier using only the features that were previously worth zero, we get an accuracy of 0,9203725261932479, which is still very high. We can conclude that the features that were not selected before were not all irrelevant in determining the number of the verb. Maybe they were not selected only because they were redundant with other selected features. When we train the classifier again and again only on the features that were not selected before, the accuracy decreases at each step before stagnating at 0,682731858750485. This means that the features become less and less relevant for determining if the verb is singular or plural. The accuracy reaches its bottom value when 259 features are used. This seems to confirm our hypothesis that only about 500 features were useful for determining the information we are interested in, because 768 (total number of features) - 259 (number of features used for the lowest accuracy) = 509 (number of features not used then).

For C = 0,01 and C = 0,1 the number of features used at each step (execution of the loop) decreases quicker then for C = 0,001 because the number of features selected (not worth zero) is higher, the complexity of the classification function allowed being higher. Just as before, the accuracy decreases at each step, while still staying relatively high for a couple of steps at the beginning. This confirms the idea that not all the features not selected initially were irrelevant.

If C is superior to 1 there are no zero parameters anymore to retrain the classifier on, so the experiment cannot be done.