CSEC 620 | Assignment 1 report

Team 4 - Nick Mangerian, Jacob Ruud, Hugo Tessier Due on 09-16-2021

This work is based on *The SMS Spam Collection* used in the study of SMS Spam Filtering (Reference (4) and (5)).

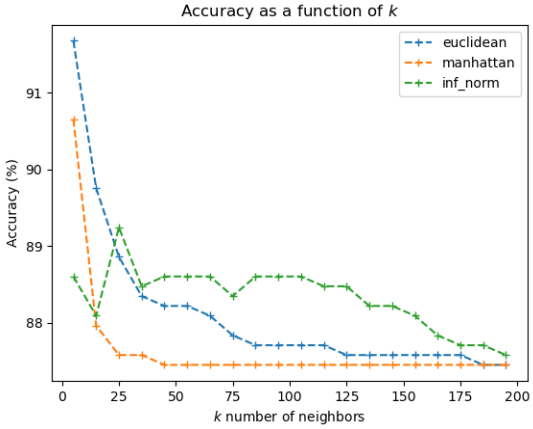
1. **How does the choice of *k* and/or distance metrics affect the performance of the nearest neighbors algorithm? Include graphs to illustrate your findings.**

The effects of *k* and/or distance metrics on the performance can be separated on two fields: on the **classifying performances** of the k-NN algorithm and on the **execution time**.

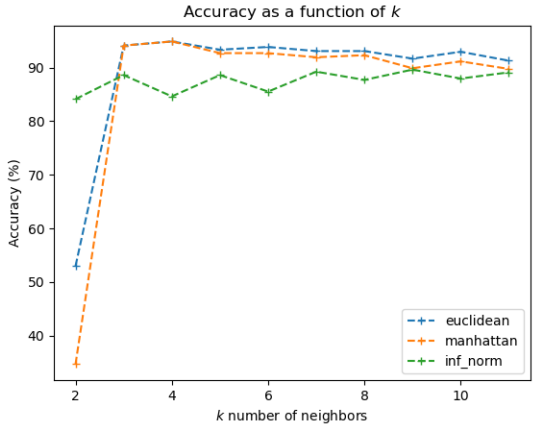
- **Effect of the distance metrics:** in this project we considered 3 different distances, the **euclidean distance**, the **Manhattan distance** and the **infinity norm distance** (Reference 1) these three norms are the most common for machine learning applications.

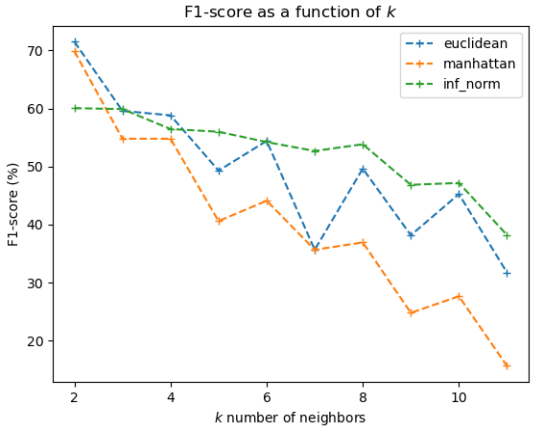
- **Effect of the choice of k:** to study the impact of the choice of *k* on the classifying performances we ran our algorithm on the same training/test sets and compared metrics as **accuracy** and **F1-Score**. We first considered a wide range of *k* and then narrowed around the best value.

We first plot the accuracy for a wide range *k* values for all the considered distances.

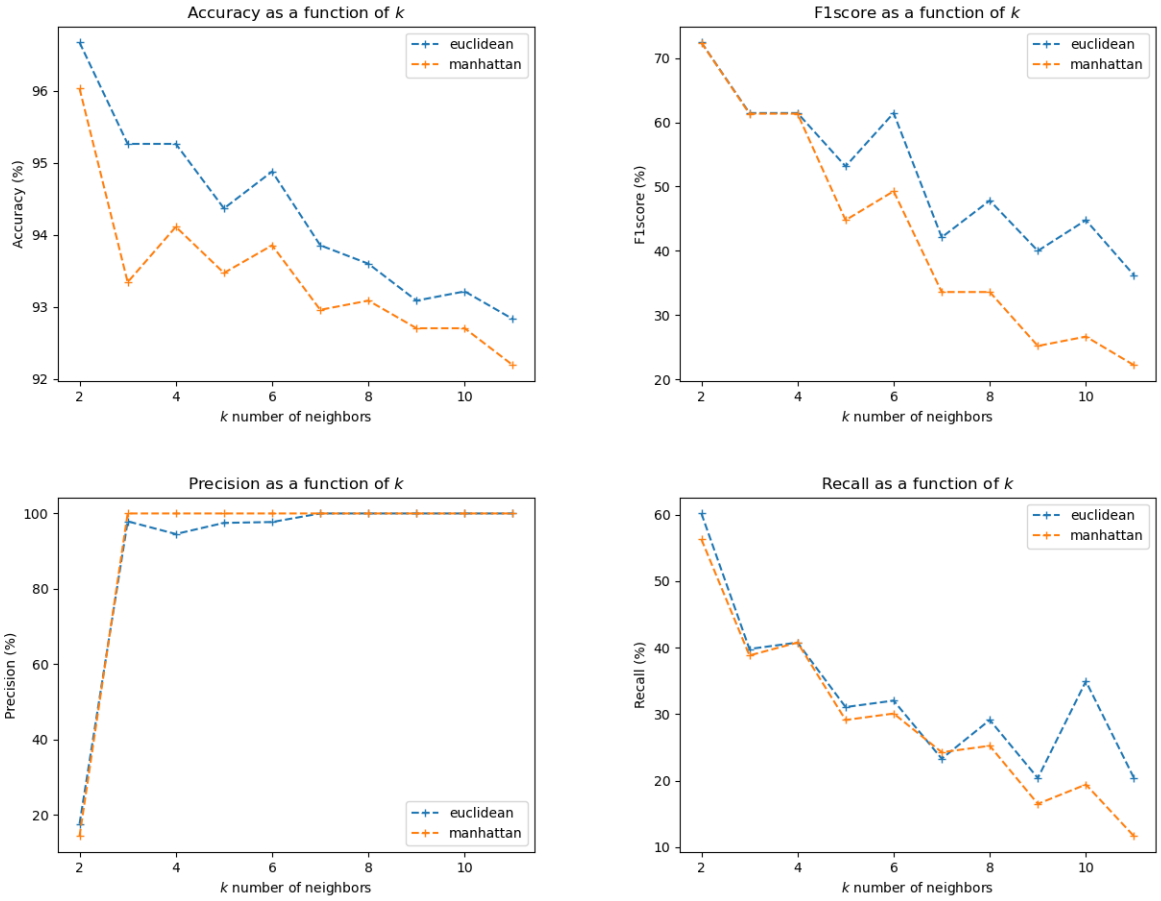


The range to consider regarding the plot is around 5 and 10 neighbors. Then, we plot the accuracy and the F1-score for this narrow range of *k*.





As we can see on the graphs, the infinity norm distance isn't the best to use for this application. We now compare the two kept distances and the number of neighbors for several scores.



Concerning the choice of distance: euclidian, Manhattan and infinity norm have the same complexity and execution times. **Euclidean distance** has better overall performances and will be kept.

Concerning the choice of *k*: the value **k=4** seems to perform better on the several scores. k=2 could be a choice but it performs more randomly as it is less aggregative. This choice doesn't have an impact on the execution time as it doesn't change the complexity of the search which is O(n). Most of the execution time is spent calculating the distances.

---

2. You have learned from the video and book that there is no training time in *k*-NN. Report your observation about this from your implemented *k*-NN.

To implement k-NN, we only had to preprocess the tokens and compute TF-IDF. There isn't any training time. All the classification part is made from the computation of the distances between the test message and the train messages. This can be observed in our algorithm because there aren't any train and predict functions but only a run function which classifies new messages.

3. Comparison:

a. Compare the accuracy of the train and test dataset for both the classifiers. Discuss if your classifier has any overfit/underfit issue.

kNN:

- Accuracy on train set = 96.28 %
- Accuracy on test set = 93.572 %

We observe overfitting issues by computing the accuracy on the train set compared to the test set. Predicting on training data biases our accuracy. At its limit, with 1 neighbor in kNN on the train set it can be analytically proven that the accuracy is always 100%.

Naive Bayes:

- Accuracy on training set = 98.685 %
- Accuracy on test set = 99.334 %

The issue is the same with this algorithm: testing on the training set has better performances but creates bias.

Remark:

We only compare the accuracy metric but the observations would be the same for all the other score metrics. Moreover, as we saw, there is a bias if we compute performance on the training set. That's the reason why we considered a training/validation for tuning different from the test set.

b. Provide the precision, recall, accuracy, and F1-score values of the k-NN and Naive Bayes classifiers.

For this question we separated the SMS dataset into train and test datasets so that the algorithms were trained on the same dataset. Then the performances had been compared between the two on the same test set so without bias. The kNN algorithm used is the best regarding the tuning made in question 1.

kNN:

Confusion matrix:

	Predicted ham	**Predicted spam**
True ham	85.356 %	3.347 %
True spam	1.375 %	9.922 %

accuracy= 95.278 %
precision= 87.831 %
recall= 74.775 %
F1-score= 0.8077858880778588

Note: The poor performance in the False Positive cell is mainly due to the ratio of 15.5 spam messages for 100 non-spam messages.

Naive Bayes:
Confusion matrix:

	Predicted ham	**Predicted spam**
True ham	86.551 %	0.598 %
True spam	0.179 %	12.672 %

accuracy= 99.223 %
precision= 98.605 %
recall= 95.495 %
F1-score= 0.9702517162471395

c. How does the performance and efficiency of the two algorithms compare? Is one better than the other? Why do you think that may be the case?

Execution time comparison (Machine properties: i7-1165G7, RAM 16Go)
time kNN: 271.724s, time Naive Bayes: 0.303s

The main difference between those two algorithms is their **complexity**: Naive Bayes is much faster. It's due to the number of distances that has to be computed in kNN and the use of TF-IDF which leads to a high-dimensional dataset (a lot of features) as each unique word is a feature in the feature vector.

Furthermore, Naive Bayes classification also has better performances regarding the question 3.b) (especially for the recall). This classifier is more efficient. For the spam detection, Naive Bayes looks more adapted because the time required to classify a SMS is important: one doesn't want to delay the message reception too much for the customer.

d. Are there techniques that can be used to improve the performance of these classifiers (e.g. improvements to text tokenization or the Naive Bayes/k-NN algorithms)?

i. ***Bonus: Compare the results of applying these techniques with the original classifiers.***

We have tried several techniques to improve the performance of these classifiers. Here are listed the improvement techniques used.

For tokenization:

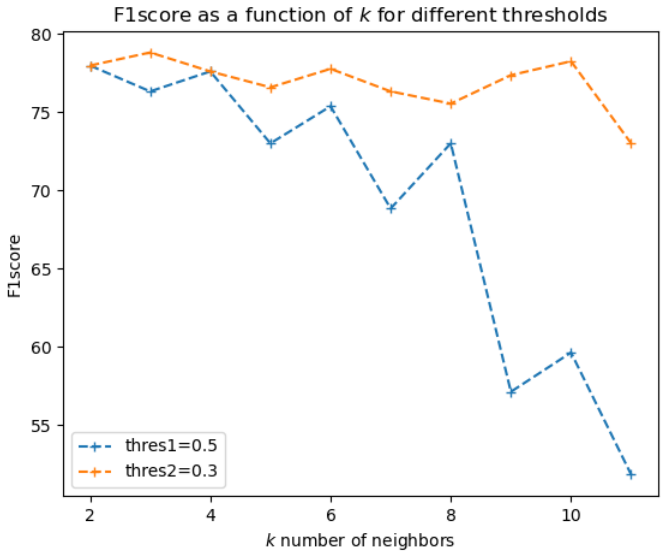
- **Lowercase words:** it allows to group words which are the same together. Thus it reduces the number of unique words and the aggregating effect improves the performances.
- **Punctuation:** we add the detection of punctuations as a token to be able to also somehow get information on the category of the SMS. For example, interrogation points help identify questions, slash characters identify URLs, hyphens can be useful for finding phone numbers. This increases the accuracy of our best classifier (Naive Bayes) by 5%.

For Naive Bayes:

- **Logarithm:** to not be limited by the machine epsilon we use addition of logarithm instead of multiplication. This increases the accuracy metric by 3%.
- **Alpha value:** as shown in question 6, we have to add an arbitrary initial value to each word in the ham and spam dictionary that we called alpha. A first option would be to set alpha=1. We tried several values from 1 to 0.001 and it appears that the performances weren't affected by the choice of alpha.

For kNN algorithm:

- **Computation of the distance:** the slowness of the algorithm is due to the computation of distances. We found a way to fasten the computation in our specific case. We first used array programming with the numpy (Reference (2)) library. Then, we also observed that the arrays made by TF-IDF contain a lot of zeros. We reduced our number of coefficients to sum in the Manhattan and Euclidean distances by only considering the non-zeros coefficients. This significantly improved the execution time of k-NN.
- **Feature selection:** the distances take so long to compute because of the large number of parameters. To fasten the computation, we did a feature selection in the preprocessing phase (before applying TD-IDF). We add a minimum threshold on the number of occurrences of a word. For instance putting the threshold at one will remove words that appear only once. Going from 1 to 2 reduces our feature vector from 7000 components to 3500 components, from 2 to 3 reduces from 3500 to 3000.
- **Cross-validation:** this method hasn't been implemented but would be efficient to be less dependent on the validation set that was randomly selected.
- **Threshold-Moving for Imbalanced Classification:** we used a method mostly known for logistic regressions models to improve the classification performances of this classifier (Reference (3)). We can use this method because our spam and ham categories aren't balanced. As the density of spam messages is lesser than non-spam, it consists in more easily classifying a message as spam by giving more weight to spams in our neighborhood. We plot the differences of the F1-score for two thresholds 0.5 and 0.3.



Putting the threshold to 0.3 instead of 0.5 improves the accuracy. We should use the ROC curve to find the best threshold.

- **Cosine distance:** trying to evaluate the performances with cosine distance could also be an improvement clue as this distance is powerful for document classification.

e. What are the shortcomings and limitations of each classifier?

For both classifiers, one limitation is that each word is taken independently, it's not able to get the overall feeling of the SMS by considering a bag of words. Also, the models are sensitive to spelling mistakes.

kNN:

- **Computation time:** one limitation of kNN is its complexity that implies long computation time.
- **Large feature vector:** the large feature vector implies that messages are easily far from each other. In our case, our dataset can have more features than messages. This leads to a well known limitation which is the **curse of dimensionality.**

---

4. Problem from Class: Suppose that the Accuracy of a malware classifier was 95%, with equal rates of false positives and false negatives (FPR = FNR = 5%). For a test dataset with 10,000 samples and 99.9% benign software, compute:

a. The expected number of benign and malicious programs in the test set.

i. $expected\ benign = 10,000 \times .999 = 9990$

ii. $expected\ malware = 10,000 \times .001 = 10$

b. The expected number of TPs, TNs, FPs, and FNs.

i. $expected\ TP = 10 * .95 = 9.5$

ii. $expected\ TN = 9990 * .95 = 9490.5$

iii. $expected\ FP = 9990 * .05 = 499.5$

iv. $expected\ FN = 10 * .05 = .5$

c. The Precision and Recall values.

i. $precision = TP \div (TP + FP) = 9.5 \div (9.5 + 499.5) = .018\ or\ 1.8\%$

ii. $recall = TP \div (TP + FN) = 9.5 \div (9.5 + .5) = .95\ or\ 95\%$

---

5. Suppose that the Accuracy of a classifier on this dataset was 93%, with equal rates of false positives and false negatives (FPR = FNR = 7%). Compute:

a. The expected number of TPs, TNs, FPs, and FNs.

i. $expected\ TP = 10 * .93 = 9.3$

ii. $expected\ TN = 9990 * .93 = 9290.7$

iii. $expected\ FP = 9990 * .07 = 699.3$

iv. $expected\ FN = 10 * .07 = 0.7$

b. The Precision and Recall values.

i. $precision = TP \div (TP + FP) = 9.3 \div (9.3 + 699.3) = .013\ or\ 1.3\%$

ii. $recall = TP \div (TP + FN) = 9.3 \div (9.3 + 0.7) = .93\ or\ 93\%$

c. **Suppose that you are an intern in the RIT ITS spam division, and you have been tasked with manually checking all the messages detected as SMS spam using this classifier to see if they are spam or not. It takes you one hour to check 100 messages. How much of your time will be spent looking at actual spam? How much will be spent looking at non-spam?**

i. $actual\ spam\ time = 9.3\ actual\ spam * (1hr/100\ messages) = 0.093hr\ or\ 5.58$

ii. $non - spam\ time = 699.3\ false\ spam * (1hr/100\ messages) = 6.9hrs$

This is due to the imbalance of the categories in the dataset. There are more non-spam than spam messages. We can observe from this example the phenomenon of the **base rate fallacy**.

---

6. **Provide the detailed calculations and results in your report for the following toy example. You are preparing a simple Bayes classifier using the following word frequency dictionaries:**

W_{Spam} = ["click": 2, "dude": 1, "prize": 3, "for": 3, "look": 1, "winner": 3]

$W_{not\ Spam}$ = ["babe": 1, "dude": 3, "look": 2]

a. **Calculate $P(w|Spam)$ and $P(w|not\ Spam)$ for every word W_{Spam} and $W_{not\ Spam}$.**

The formulas used are:

$P(w \mid Spam) = \frac{number\ of\ w\ in\ spam}{total\ number\ of\ words\ in\ spam}$, $P(w \mid not\ Spam) = \frac{number\ of\ w\ in\ not\ spam}{total\ number\ of\ words\ in\ not\ spam}$

word	*click*	*dude*	*prize*	*for*	*look*	*winner*	*babe*
$P(word \mid Spam)$	2/13	1/13	3/13	3/13	1/13	3/13	0/13 = 0
$P(word \mid not\ Spam)$	0	3/6	0	0	2/6	0	1/6

b. **Using your results from *i.* compute $P(Spam|W)$ and $P(not\ Spam|W)$ for the following SMS when assuming that messages are spam 10% of the time. The SMS is *"dude! dude! look!"*.**

$P(Spam|W) = P(Spam) \times P(dude|Spam) \times P(dude|Spam) \times P(look|Spam)$

$= \frac{1}{10} \times \frac{1}{13} \times \frac{1}{13} \times \frac{1}{13}$

$= 0.00004551661$

$P(not\ Spam|W) = P(not\ Spam) \times P(dude|not\ Spam) \times P(dude|not\ Spam) \times P(look|not\ Spam)$

$= \frac{9}{10} \times \frac{3}{6} \times \frac{3}{6} \times \frac{2}{6}$

$= 0.075$

i. ***Is the message spam or not spam?***

$P(not\ Spam|W) > P(Spam|W)$, so it is not a spam message.

c. Do the same calculations for the known spam: *"winner babe! click for prize".*

$$P(Spam|W) = P(Spam) \times P(winner|Spam) \times P(babe|Spam) \times P(click|Spam) \times P(for\ |Spam) \times P(prize|Spam)$$
$$= \frac{1}{10} \times \frac{3}{13} \times \frac{0}{13} \times \frac{2}{13} \times \frac{3}{13} \times \frac{3}{13}$$
$$= 0$$

$$P(not\ Spam|W) = P(not\ Spam) \times P(winner|not\ Spam) \times P(babe|not\ Spam) \times P(click|not\ Spam) \times P(for\ |not\ Spam) \times P(prize|not\ Spam)$$
$$= \frac{9}{10} \times \frac{0}{6} \times \frac{2}{6} \times \frac{0}{6} \times \frac{0}{6} \times \frac{0}{6}$$
$$= 0$$

i. What does the classifier say about this message? Why?

The classifier says that the message has a zero probability of being spam or not spam. This is because words were used that exist only in the other set.

ii. Consider how this issue may be solved when building your classifier.

To solve this issue, you would add one to each of the words for every word in both spam and not spam sets to both spam and not spam sets. We did this and was able to get a prediction of spam. For every word we added 1 in order to not have value of 0 for probabilities for either set and therefore the frequencies are shown below:

Frequency for spam:

W_{Spam} = ["click": 3, "dude": 2, "prize": 4, "for": 4, "look": 2, "winner": 4, "babe":1]

Frequency for not spam:

$W_{not\ Spam}$ = ["click": 1,"dude": 4, ", "prize": 1, "for": 1, look": 3,"winner": 1,"babe": 2]

word	*click*	*dude*	*prize*	*for*	*look*	*winner*	*babe*
$P(word \mid Spam)$	3/20	2/20	4/20	4/20	2/20	4/20	1/20 = 0
$P(word \mid not\ Spam)$	1	4/13	1/13	1/13	3/13	1/13	2/13

Probability calculation

$$P(Spam|W) = P(Spam) \times P(winner|Spam) \times P(babe|Spam) \times P(click|Spam) \times P(for\ |Spam) \times P(prize|Spam)$$
$$= \frac{1}{10} \times \frac{4}{20} \times \frac{1}{20} \times \frac{3}{20} \times \frac{4}{20} \times \frac{4}{20}$$
$$= 0.000006$$

$$P(not\ Spam|W) = P(not\ Spam) \times P(winner|not\ Spam) \times P(babe|not\ Spam) \times P(click|not\ Spam) \times P(for\ |not\ Spam) \times P(prize|not\ Spam)$$

$= \frac{9}{10} \times \frac{1}{13} \times \frac{2}{13} \times \frac{1}{13} \times \frac{1}{13} \times \frac{1}{13}$
$= 0.00000484792$

Probability is higher for spam. Therefore, it is spam.

References

1) "Norm (mathematics)", Wikipedia, Wikimedia Foundation, 16 September 2021, https://en.wikipedia.org/wiki/Norm_(mathematics)#Maximum_norm_(special_case_of:_infinity_norm,_uniform_norm,_or_supremum_norm).
2) Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2
3) "A Gentle Introduction to Threshold-Moving for Imbalanced Classification", machinelearningmastery, Jason Brownlee, 5 January 2021 https://machinelearningmastery.com/threshold-moving-for-imbalanced-classification/
4) Almeida, T.A., Gómez Hidalgo, J.M., Yamakami, A. Contributions to the study of SMS Spam Filtering: New Collection and Results. Proceedings of the 2011 ACM Symposium on Document Engineering (ACM DOCENG'11), Mountain View, CA, USA, 2011.
5) SMS Spam Collection, Tiago Agostinho de Almeida and José María Gómez Hidalgo http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/