Amirkabir University of Technology
(Tehran Polytechnic)

Report of the Project about Spam detection in Persian

Professor:
Dr. M.Akbari (akbari.ma@aut.ac.ir)
Teaching Assistants:
A.Malekzade (malekzadeh@ieee.org) , Y.Ommi(yassi.ommi@gmail.com)

Student:
Mohammadreza Ardestani **(ardestani.zr@gmail.com)**

5, Dec, 2020

0) **Introduction**

Phase 1) **Preprocessing the data and preparing it**

Phase 2) **Best feature selection with Chi-square method**

Phase 3) **Finding Cosine-similarity and tf-idf**

Phase 4) **Implementing KNN from scratch**

Phase 5) **Implementing Naïve Bayes Method**

Phase 6) **Calculating Precision, Recall, F1, Confusion Matrix**

6.1) Finding the measures for KKN, KKN with K-Best-Features ,& Naive Bayes

6.2) Comparing KKN & KKN with k-Best

6.3) Comparing KKN ,& Naive Bayes

(I've implemented all of **Bonus questions**)

0) Introduction

This report will be concise but thorough. If you need more detail, please contact me.

How to run:

I. At first you should have been installed all required libraries and Jupyter notebook.
II. Create a folder & paste Ass2_spamDetectionInPersainLang.ipynb' and ' stopwords.txt ' and 'emails.folder' .
III. At the address bar write “cmd” then click “Enter”.
IV. At the Cmd console type “jupyter notebook” and then “Enter”.
V. Open Jupyter sourceCode and run each cell in order
VI. **Notice some cells take a few second to run completely**

Pre analysis of Input data and deciding which part are useful for model:

Before using Test set I have remover all words that are not in trainset :

in cosine similarity fucntion)

```
In [13]: def lenght(listOfDics):
             # this fucn get a list of dictionaries and calculate total number of words in all dics
             s = 0
             for dic in listOfDics:
                 s += len(dic)
             return s

         def unknownRemover(testDic ,DFA):
             # we go through all dictionaries in testdic and remover unknown words
             # notice testDic is a list of dics and DFA is a dic
             for dic in testDic:
                 CFR = [] #CandidateForRemoving
                 for k,v in dic.items():
                     if k not in DFA:
                         CFR.append(k)
                 for k in CFR:
                     del dic[k]
```

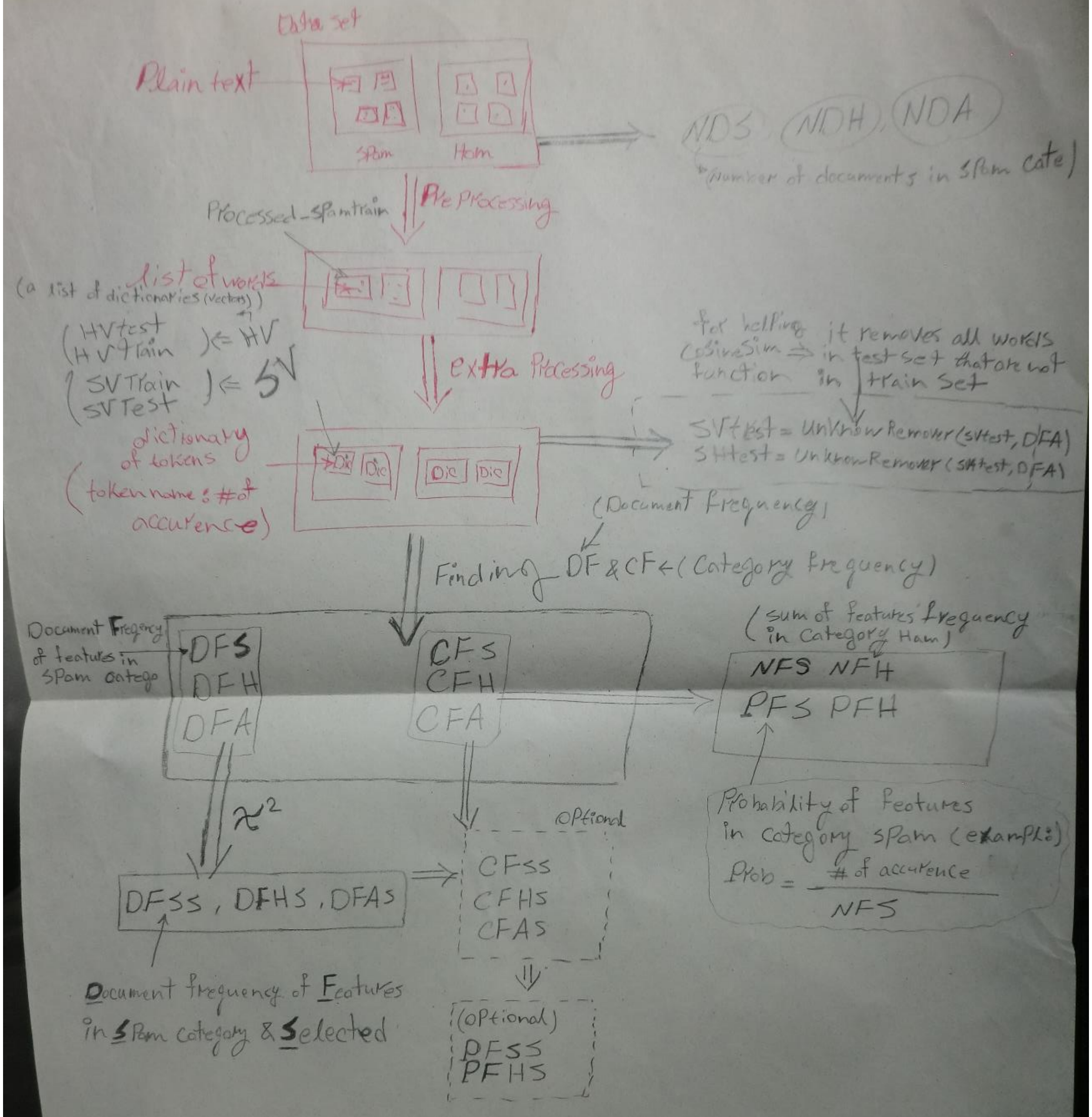
```
In [14]: print("total words in SVTest and SHTest before removing unkowns",lenght(SVTest),lenght(HVTest))
         unknownRemover(SVTest,DFA)
         unknownRemover(HVTest,DFA)
         print("total words in SVTest and SHTest after removing unkowns",lenght(SVTest),lenght(HVTest))

total words in SVTest and SHTest before removing unkowns 26033 16735
total words in SVTest and SHTest after removing unkowns 23654 14783
```

And also we find our whether there is any empty email or not. And there is not any empty email.

Phase 1) **Preprocessing the data and preparing it**

We have a long road in Preprocessing and further processing. To sum up this long story look at the following graph of preprocessing:


Data set
Plain text
Spam
Ham
NDS NDH NDA
*(number of documents in Spam cate)
Pre Processing
Processed_Spamtrain
list of words
(a list of dictionaries (vectors))
(HVtest HVTrain) ⇐ HV
(SVTrain SVTest) ⇐ SV
extra Processing
for helping CosineSim function ⇒ it removes all words in test set that are not in train set
SVtest = UnknowRemover(SVtest, DFA)
SHtest = UnknowRemover(SHtest, DFA)
dictionary of tokens
(token name : # of accurence)
Dic Dic Dic Dic
(Document Frequency)
Finding DF & CF ← (Category Frequency)
Document Frequency of features in Spam catego
DFS
DFH
DFA
CFS
CFH
CFA
(sum of features frequency in Category Ham)
NFS NFH
PFS PFH
Probability of features in category spam (example:)
Prob = # of accurence / NFS
χ^2
Optional
DFSS, DFHS, DFAS
CFSS
CFHS
CFAS
(optional)
PFSS
PFHS
Document frequency of Features in Spam category & Selected

Phase 2) **Opting the best features with CHI-SQUARE method**

Following terminology describes CHI-Square method and we how to calculate it.

$$x^2(t_k, C_i) = \frac{N * (AD - CB)}{(A+C)*(B+D)*(A+B)*(C+D)}$$

C_i \ t_k	1	0
1	A	C
0	B	D

A = #(number) of tweets in C_i category which have the term "t_k"

C = # of tweets in C_i Cate which don't have the term "t_k"

B = # of tweets which are not in C_i & have the "t_k"

D = # of tweets which are not in C_i & don't have "t_k"

N = # of all tweets in trainset document

After we calculate Chi-square for all words, we need to pick up some of them that have more value (or more correlation).

In Final_Bag of features we select those words which have more than 0.1 Chi-square value.

```
In [18]: print(important_spam_words)
```

```
[('پست', 23.81), ('اموز', 23.619), ('استفاده', 18.468), ('ارسال', 10.115), ('سا', 8.952), ('کلیک', 8.322), ('نر', 8.034), ('باشد', 7.99), ('افزار', 7.909), ('مو', 7.192), ('زیبا', 7.137), ('مجموعه', 6.734), ('ایمیل', 6.456), ('رو', 6.201), ('صور', 5.433), ('افزا', 5.416), ('دارا', 5.253), ('فر', 4.955), ('اموزش', 4.899), ('دکمه', 4.888), ('فرو', 4.833), ('د', 4.794), ('دستگاه', 4.789), ('منزل', 4.751), ('شده', 4.724), ('مشاهده', 4.715), ('ساع', 4.663), ('طراح', 4.65), ('العاده', 4.63), ('نمایید', 4.621), ('فوق', 4.611), ('توانید', 4.61), ('قابل', 4.575), ('شامل', 4.521), ('کالا', 4.515), ('زب', 4.484), ('هزینه', 4.44), ('همراه', 4.419), ('شود', 4.394), ('بگیرید', 4.363), ('کسب', 4.363), ('رایگ', 4.318), ('نیاز', 4.303), ('س', 4.3), ('سفار', 4.24), ('هدیه', 4.224), ('بالا', 4.183), ('شماره', 4.167), ('فروشگاه', 4.143), ('مناسب', 4.09), ('رنگ', 4.07), ('کودک', 4.066), ('سیس', 4.066), ('میباشد', 4.052), ('سپس', 4.028), ('اینترنت', 4.028), ('نظیر', 4.0), ('بسته', 3.997), ('پوس', 3.971), ('وب', 3.968), ('شوید', 3.961), ('فیل', 3.958), ('جذاب', 3.952), ('فروشنده', 3.899), ('عدد', 3.883), ('بپردازید', 3.883), ('کاربر', 3.877), ('دیجیتال', 3.876), ('پکیج', 3.861), ('عینک', 3.846), ('الکترونیک', 3.843), ('کلیه', 3.839), ('اندا', 3.817), ('اورجینال', 3.795), ('پشتیبان', 3.795), ('درب', 3.79), ('لاغر', 3.774), ('کیف', 3.773), ('مجوز', 3.766), ('شیک', 3.752), ('درامد', 3.75), ('وزن', 3.742), ('کاه', 3.74), ('دریاف', 3.732), ('ویندوز', 3.731), ('پاور', 3.724), ('بازار', 3.716), ('سایز', 3.711), ('انواع', 3.71), ('مدل', 3.7), ('قارچ', 3.7), ('باز', 3.695), ('دند', 3.683), ('انلاین', 3.676), ('استاندارد', 3.676), ('دیدن', 3.666), ('دهید', 3.666), ('پیشرفته', 3.663), ('طبیع', 3.66), ('مگنا', 3.65), ('ایکس', 3.65), ('فارس', 3.646), ('بازاریاب', 3.643), ('موسیق', 3.636), ('بازیکن', 3.623), ('حرفه', 3.622), ('قد', 3.622), ('تناسل', 3.617), ('اتو', 3.617), ('مود', 3.617), ('روش', 3.615), ('میتوانید', 3.615), ('تعمیر', 3.61), ('بذر', 3.61), ('تبلیغات', 3.606), ('انزال', 3.604), ('وجه', 3.598), ('تضمین', 3.597), ('مامور', 3.584), ('پنل', 3.584), ('تخفیف', 3.58), ('تناسب', 3.58), ('ناخن', 3.578), ('بالانس', 3.578), ('دنیا', 3.571), ('سانت', 3.571), ('تبل', 3.571), ('تصاویر', 3.568), ('پروژه', 3.568), ('صدا', 3.562), ('اینستایلر', 3.559), ('حسابدار', 3.559), ('تیشر', 3.559), ('بدن', 3.555), ('کامل', 3.554), ('فایل', 3.554), ('معتبر', 3.554), ('دانلود', 3.551), ('مصرف', 3.548), ('دایم', 3.546), ('چرب', 3.546), ('مدفون', 3.546), ('گرافیک', 3.546), ('بورس', 3.545), ('خستگ', 3.54), ('لپ', 3.54), ('مژه', 3.54), ('ابتدا', 3.537), ('مخصوص', 3.537), ('هزارتو', 3.534), ('گزینه', 3.527), ('چین', 3.521), ('اتوماتیک', 3.521), ('مواد', 3.518), ('اسپر', 3.515), ('صاف', 3.515), ('متولدین', 3.515), ('شامپو', 3.515), ('صدف', 3.515), ('شارژر', 3.509), ('دکوراسیون', 3.509), ('کمپان', 3.503), ('فتوشاپ', 3.503), ('عطر', 3.503), ('گن', 3.497), ('رزتا', 3.497), ('استون', 3.497), ('گلد', 3.497), ('دارید', 3.495), ('هنگا', 3.494), ('طلا', 3.491), ('ماساژ', 3.49), ('معجزه', 3.49), ('چاق', 3.484), ('سایت', 3.484), ('هاله', 3.484), ('نفیس', 3.484), ('انگ', 3.484), ('هزار', 3.479), ('گروه', 3.479), ('قوط', 3.478), ('دوبله', 3.478), ('دستبند', 3.478), ('امواج', 3.478), ('لواز', 3.478), ('لیزر', 3.478), ('اکان', 3.472), ('سرامیک', 3.472), ('باف', 3.472), ('پد', 3.472), ('زل', 3.472), ('ارشیو', 3.472), ('جادو', 3.467), ('کانادا', 3.467), ('مکمل', 3.466), ('اشانتیون', 3.466), ('مایع', 3.466), ('امروزه', 3.466), ('گردنبند', 3.466), ('رادیو', 3.466)]
```

Important ham words

Phase3) **Finding Cosine-similarity and tf-idf**

Simply we use following formulas to find out Tfidf value or CosineSimilarity

$$Sim(A,B) = \frac{AB}{|A||B|} = \frac{x_{1A}x_{1B} + x_{2A}x_{2B} + \dots}{\sqrt{x_{1A}^2 + x_{2A}^2 + \dots} + \sqrt{x_{1B}^2 + x_{2B}^2 + \dots}} \qquad (۱)$$

$$\text{Score}_{\text{tf-idf}}(\hat{e}, e_i) = \sum_{i=1}^{T} \text{tf}(w_i, \hat{e})\text{idf}(w_i) \qquad (۲)$$

در عبارت ۲ مقادیر $\text{tf}(w_i, \hat{e})$ و $\text{idf}(w_i)$ به شکل زیر محاسبه می‌گردند:

$$\text{tf}(w_i, \hat{e}) = \log(\text{count}(w_i, \hat{e}) + 1) \qquad (۳)$$

$$\text{idf}(w_i) = \log(\frac{n}{\text{df}(w_i)}) \qquad (۴)$$

```
def cosine(dic10 , dic20 , selected):
    # dic1 is query and dic2 is already exist in train set and Selected is a dic of important words
    # if selected is a null dictionary, it means we should consider all words in our trainSet
    s = 0 # score
    dic1 = dic10 # prevent damaging the Train/test set documents by removing words (optimize later)
    dic2 = dic20 # prevent damaging the Train/test set documents by removing words (optimize later)
    if(selected=={}):
        # just we check size of dics
        if(len(dic1)==0 or len(dic2)==0 ):# preventing divide by 0
            #print("null doc!")
            #print("dic1",dic1)
            #print("dic2",dic2)
            return 0
    else:
        unknownRemover([dic1],selected)
        unknownRemover([dic2],selected)
        if(len(dic1)==0 or len(dic2)==0 ): # preventing divide by 0
            return 0
    norm1 = normCal(dic1)  # |A|
    norm2 = normCal(dic2)  # |A|
    if(len(dic1)==0 or len(dic2)==0 ): # Remove this later(i.e we shouldn't have hull dic1 or dic2 as input)
        return 0
    dotProduct = dot(dic1,dic2) # A.B
    s = (dotProduct/(norm1*norm2)) # Cosine of A&B
    return s
```

```python
def tfIdf(dic10 , dic20 ,selected):
    # dic1 is query and dic2 is already exist in train set and Selected is a dic of important words
    # if selected is null, it means we should consider all words in our trainSet
    s = 0 # score
    dic1 = dic10 # prevent damaging the Train/test set documents by removing words (optimize later)
    dic2 = dic20 # prevent damaging the Train/test set documents by removing words (optimize later)
    if(selected =={}):
        pass
    else:
        unknownRemover([dic1],selected)
        unknownRemover([dic2],selected)


    for k,v in dic1.items():
        if k in dic2:  #Check if Key Exists in Dic2
            s += math.log(dic2[k]+1) * math.log(NDA/DFA[k])    # tf*idf

    return s

# optional:
def distance(dic1,dic2): # Euclidean distance
    pass
```

Phase 4) **Implementing KNN from scratch**

```python
In [29]: def KNN(dic1, flag ,k, selected):
    # input=(queryDocument,function is used for finding similarity ,number of neighbors, selected features)
    # type of imputs (dictionary, num , num , dictionary)
    # we have access to our train set, So we don't pass train set
    # selected feature or in other word, most important words
    # if flag is 0   we use Cosine func
    # if flag is 1 we use tfIdf

    neighbors = [(0,0)]*k # k nearest neighbors and their similarity value
    # ( name of category of neighbor , similarity value)

    if(flag==0):
        # we use cosine similarity function

        # passing all dics in SVTrain
        for dic2 in SVTrain:
            neighAdder(neighbors,k, 1 , cosine(dic1 , dic2 , selected)  )
        # passing all dics in SVTrain
        for dic2 in HVTrain:
            neighAdder(neighbors,k, 0 , cosine(dic1 , dic2 , selected)  )



    if(flag==1):
        # we use tfIdf similarity function

        # passing all dics in SVTrain
        for dic2 in SVTrain:
            neighAdder(neighbors,k, 1 , tfIdf(dic1 , dic2 , selected)  )
        # passing all dics in SVTrain
        for dic2 in HVTrain:
            neighAdder(neighbors,k, 0 , tfIdf(dic1 , dic2 , selected)  )


    # at the end we return 0 (Ham) or 1 (Spam)
    return bestNeighbor(neighbors)
```

Simply we get a query and we find its similarity comparing to all documents in the train set and then we report the nearest class.

Phase 5) **Implementing Naïve Bayes from scratch**

$$\text{Best Category for document}_j = \max_{i \in \text{Categories}} \{ P(c_i | d_j) \}$$

Bayes rule: $P(A|B) = \dfrac{P(A \wedge B)}{P(B)}$, $P(B|A) = \dfrac{P(B \wedge A)}{P(A)}$ $\Rightarrow$ $\underbrace{P(A|B) = \dfrac{P(B|A) \cdot P(A)}{P(B)}}_{*}$

So:

$$P(c_i|d) = \frac{P(c_i \wedge d)}{P(d)} \overset{*}{=} \frac{P(d|c_i) \cdot P(c_i)}{P(d)} \overset{**}{=} \frac{P(w_1, ..., w_n | c_i) \frac{|c_i|}{|N|}}{P(d)}$$

$$\overset{***}{\propto \simeq} P(w_1, ..., w_n | c_i) \cdot \frac{|c_i|}{|N|} = \frac{|c_i|}{|N|} \cdot \left(P(w_1|c_i) \dots P(w_n|c_i) \right)$$

** : withaut damaging accuracy, we assume all words in a damcument are Independent.

$P(c_i) = \dfrac{|\text{all tweets in Category } c_i|}{|\text{all tweets}|}$

*** : since $P(d)$ is a fixed & nonchanging value & it doesn't depend on any specific Category, so we can omit $P(d)$ from formula

Finally:

$$\text{Best Category for "}d_j\text{"} = \max_{i \in \text{cates}} \{ (P(c_i) \cdot P(w_1, ..., w_n | c_i)) \}$$

More detail:

{All words are independent}

$$P(w_1, \dots, w_n \mid C_i) = \prod_{1 \le j \le n} P(w_j \mid C_i)$$

$$P(w_j \mid C_i) = \frac{\#\text{of occurence of word}_j \text{ in Category } C_i}{\text{number of all words in } C_i}$$

*But "w_j" might don't exist in Category C_i

For handeling above Problem we smooth the formula by one ⇒

$$P(w_j \mid C_i) = \frac{\#\text{ of occurence of } w_j \text{ in } C_i + 1}{\text{number of all words in Category } C_i + |F|}$$

$$|F| = \#\text{ of all features in category } C_i$$

Based on the knowledge that we have gotten from last phase, now we can design a classifier. In order to do that, at first we find the value of all words in our bag of words given corresponding category ($P_{w_j \in C_i}(w_j \mid c_i)$).

We store the result in Prob_P , Prob_N , Prob dictionaries.

Prob_P is dictionary for selected features in category positive and its probability given class "positive". And same description for Prob_N that is a dictionary for class "negative" and Prob is related to class "neutral".

Classifier function gives a string and then it calculate its probability regarding to each category that we have and then return the most likely category.

```
## Naive Bayse Classifier :
def Classifier(string):
    CleanList = TextCleaner(string)#clean list of words
    resultDic={}
    resultDic["positive"] = Probability(CleanList, "positive")
    resultDic["negative"] = Probability(CleanList, "negative")
    resultDic["neutral"] = Probability(CleanList, "neutral")
    resultDic = sorted(resultDic.items(), key=lambda x: x[1], reverse=True)#reultDic is no longer a dic (It's a list)
    return resultDic[0][0]
```

sorting the result and then returning the cate's name for the greatest probability

Phase 6) **Calculating Precision, Recall, F1, Confusion Matrix**

Matrix:

	correct	not correct
selected	tp	fp
not selected	fn	tn

$$P = \text{Precision} = \frac{tp}{tp+fp}\ ,\ R = \text{Recall} = \frac{tp}{tp+fn}\ ,\ F_1 = \frac{2*PR}{P+R}$$

5.1) Naive Bayes

```
# result for  classifying Spam testset by using Bayes
SVTestBayes = BayesList(SVTest,{})
# result for  classifying Ham testset by using Bayes
HVTestBayes = BayesList(HVTest,{})
# print(SVTestBayes)
# print(HVTestBayes)
temp_ac, temp_pre, temp_recall, temp_f1 = confusion_matrix(SVTestBayes, HVTestBayes)
acc.append(temp_ac)
pre.append(temp_pre)
recall.append(temp_recall)
f1.append(temp_f1)
```

Confusion Matrix	Predicted: Spam	Predicted: Ham
Actual: Spam	163	37
Actual: Ham	2	198

5.2) KNN cosine on all words

In [35]:
```
# result for  classifying Spam testset by using KNN cosine and using All words
SVTestKNNcosine = KNNList(SVTest, 0 , 3 , {})
# result for  classifying Ham testset by using KNN cosine and using All words
HVTestKNNcosine = KNNList(HVTest, 0 , 3 , {})
# print(SVTestKNNcosine)
# print(HVTestKNNcosine)
temp_ac, temp_pre, temp_recall, temp_f1 = confusion_matrix(SVTestKNNcosine, HVTestKNNcosine)
acc.append(temp_ac)
pre.append(temp_pre)
recall.append(temp_recall)
f1.append(temp_f1)
```

Confusion Matrix	Predicted: Spam	Predicted: Ham
Actual: Spam	190	10
Actual: Ham	24	176

5.3) KNN tf-idf on all words

In [36]:
```
# result for  classifying Spam testset by using KNN tf-idf and using All words
SVTestKNNtfIdf = KNNList(SVTest, 1 , 3 , {})
# result for  classifying Ham testset by using KNN tf-idf and using All words
HVTestKNNtfIdf = KNNList(HVTest, 1 , 3 , {})
# print(SVTestKNNtfIdf)
# print(HVTestKNNtfIdf)
temp_ac, temp_pre, temp_recall, temp_f1 = confusion_matrix(SVTestKNNtfIdf, HVTestKNNtfIdf)
acc.append(temp_ac)
pre.append(temp_pre)
recall.append(temp_recall)
f1.append(temp_f1)
```

Confusion Matrix	Predicted: Spam	Predicted: Ham
Actual: Spam	195	5
Actual: Ham	58	142

5.4) KNN cosine on important words ¶

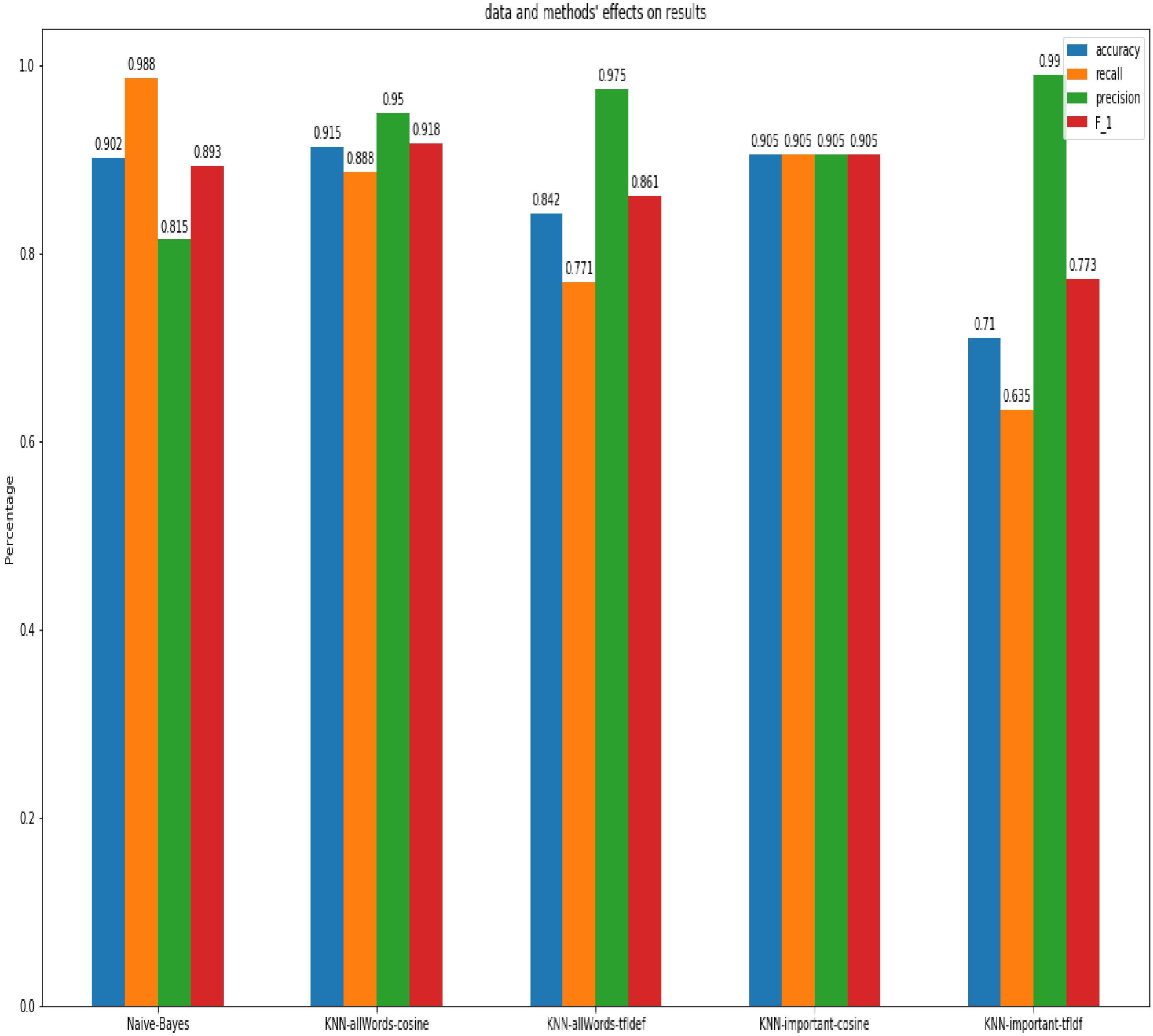
In [37]:
```
# result for classifying Spam testset by using KNN cosine and selected (important) words
SVTestKNNCosineSelected = KNNList(SVTest, 0 , 3 , DFAS)
# result for classifying Ham testset by using KNN cosine and selected (important) words
HVTestKNNCosineSelected = KNNList(HVTest, 0 , 3 , DFA)
# print(SVTestKNNCosineSelected)
# print(HVTestKNNCosineSelected)
temp_ac, temp_pre, temp_recall, temp_f1 = confusion_matrix(SVTestKNNCosineSelected, HVTestKNNCosineSelected)
acc.append(temp_ac)
pre.append(temp_pre)
recall.append(temp_recall)
f1.append(temp_f1)
```

Confusion Matrix	Predicted: Spam	Predicted: Ham
Actual: Spam	181	19
Actual: Ham	19	181

5.5) KNN tf-idf on important words

[38]:
```
# result for  classifying Spam testset by using KNN tf-idf and selected (important) words
SVTestKNNtfIdfSelected = KNNList(SVTest, 1 , 3 , DFAS)
# result for  classifying Ham testset by using KNN tf-idf and selected (important) words
HVTestKNNtfIdfSelected = KNNList(HVTest, 1 , 3 , DFA)
# print(SVTestKNNtfIdfSelected)
# print(HVTestKNNtfIdfSelected)
temp_ac, temp_pre, temp_recall, temp_f1 = confusion_matrix(SVTestKNNtfIdfSelected, HVTestKNNtfIdfSelected)
acc.append(temp_ac)
pre.append(temp_pre)
recall.append(temp_recall)
f1.append(temp_f1)
```

Confusion Matrix	Predicted: Spam	Predicted: Ham
Actual: Spam	198	2
Actual: Ham	114	86



This bar chart can summarize everything and presents the power of each method.

We can see that KNN using all words with Cosine similarity measure has the best accuracy in the average .

Thanks for your time.