ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА
(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем
(повна назва кафедри (предметної, циклової комісії))

# Дипломна робота

на здобуття ступеня вищої освіти «магістр».

на тему Дослідження і розробка системи виявлення деяких різновидів плагіату

Research and development of the system of detection of some varieties of plagiarism

Виконав: студент денної форми навчання
спеціальності 123 Комп'ютерна інженерія
Гафар Абдула І. Х.

Керівник доц., к.т.н Пенко В. Г.

Рецензент доц., к.т.н Волощук Л. А.

Рецензент доц., к.т.н Зоріло В. В.

Рекомендовано до захисту:

Протокол засідання кафедри

№_____ від «___»____2018р.

Завідувач кафедри

_____ Є.В. Малахов

(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК № ___

протокол №___ від «___» ____2018р.

Оцінка _____

(за 4-х бальною шкалою, за шкалою ECTS, бал.)

Голова ЕК

_____ О.О. Арсірій

(підпис) (прізвище, ініціали)

Одеса – 2018

# АНОТАЦІЯ

У дипломній роботі розробляється тема «Дослідження і розробка системи виявлення деяких різновидів плагіату».

Мета роботи - дослідження різних практичних підходів при побудові автоматизованих систем виявлення мультимовного плагіату. Розглянуто кілька варіантів побудови класифікаційних моделей (фільтрів), що базуються на різних варіантах компактного представлення текстів і слів: фільтр на основі закону Зіпфа, фільтр на основі розподілу високочастотних слів, фільтр на основі близькості векторного уявлення word2vec найбільш популярних слів. Запропоновані моделі реалізовані за допомогою процедури машинного навчання для досягнення найкращих показників внутрішніх параметрів. Для реалізованих моделей отримані оцінки класифікаційних метрик на невеликому тестовому корпусі текстів.

Для підвищення ефективності виявлення плагіату запропонований метод комбінованого застосування декількох фільтрів з урахуванням показників класифікаційних метрик.

# ABSTRACT

In this work the theme "Research and development of the system of detection of some varieties of plagiarism" was developed.

The purpose of the work is the study of various practical approaches in the construction of automated systems for identifying multilingual plagiarism. Several variants of building classification models (filters) based on different variants of compact text and word representation are considered: a filter based on Zipf's law, a filter based on the distribution of high-frequency words, a filter based on similarity of the word2vec vector representation of the most popular words. The proposed models are implemented using a machine learning procedure to achieve the best indicators of internal parameters. For implemented models, estimates of classification metrics are obtained on a small test corpus of texts.

To improve the efficiency of plagiarism detection, a method for the combined use of several filters with regard to the indicators of classification metrics has been proposed.

# АННОТАЦИЯ

В дипломной работе разрабатывается тема «Исследование и разработка системы обнаружения некоторых видов плагиата».

Цель работы – исследование различных практических подходов при построении автоматизированных систем выявления мультиязычного плагиата. Рассмотрено несколько вариантов построения классификационных моделей (фильтров), базирующихся на различных вариантах компактного представления текстов и слов: фильтр на основе закона Зипфа, фильтр на основе распределения высокочастотных слов, фильтр на основе близости векторного представления word2vec наиболее популярных слов. Предложенные модели реализованы с помощью процедуры машинного обучения для достижения наилучших показателей внутренних параметров. Для реализованных моделей получены оценки классификационных метрик на небольшом тестовом корпусе текстов.

Для повышения эффективности выявления плагиата предложен метод комбинированного применения нескольких фильтров с учетом показателей классификационных метрик.

ЗМІСТ

# ВСТУП

Характерною тенденцією розвитку сучасного світу є зміна пріоритетів між матеріальними і інформаційними цінностями. Важливо не стільки володіти матеріальними ресурсами, скільки вміло керувати ними, що має на увазі успішне виконання складних інформаційних процесів. У цьому сенсі слід враховувати наступні тенденції розвитку інформаційного світу:

- зростання обсягів інформації, що використовуються в процесі реалізації людиною своїх функцій;

- дедалі більша доступність цієї інформації. Це має на увазі не тільки фізичну доступність інформаційних ресурсів, але і досить досконалі і зручні засоби їх пошуку;

- підвищення цінності інформації. У цьому сенсі важливо відзначити і розширення діапазону сфер людської діяльності, в яких інформація має важливе значення.

Не дивно, що така важлива субстанція як інформація в цих умовах набуває характерні особливості товару, тобто стає об'єктом товарно-грошових відносин. У цьому сенсі важливо забезпечити інформацію властивостями, які характерні для товару і дозволяють їй повноцінно брати участь в таких відносинах.

Очевидно, що однією з таких властивостей є ставлення приналежності інформації як товару до його виробнику (автору). Однак звичайні методи визначення власника в даному випадку не можуть застосовуватися через нематеріальний характер інформації. Незважаючи на труднощі, що виникають при визначенні власника інформації, в розглянутій нами сфері існує стійке поняття «плагіат», яке позначає порушення такого ставлення власності.

Те, що для традиційних (матеріальних) видів товарів можна впевнено кваліфікувати як крадіжку, для об'єктів інформаційного характеру часто має більш розпливчатий характер. Наприклад, інформаційний ресурс, за

змістовими ознаками має яскраво виражені риси плагіату, може кваліфікуватися як пародія або як продукт, схожість якого з оригіналом виникло в результаті випадкового збігу. Несумлінне використання таких прийомів може призводити до засудження морального характеру. Однак при зростанні цінності інформаційного ресурсу в подібних ситуаціях актуальним стає застосування до порушників заходів юридичного характеру.

Таким чином, для вирішення проблем, пов'язаних зі ставленням власності щодо інформаційних ресурсів, виникає необхідність в розвитку засобів різного характеру:

- створення засобів об'єктивного визначення ступеня запозичення;
- створення відповідної юридичної підтримки.

Вирішення цих завдань має суттєві особливості для різних типів інформаційних ресурсів. Наприклад, для визначення ступеня запозичення аудіоматеріалів, зображень і текстової інформації, очевидно, потрібно використовувати дуже різні підходи і, відповідно, зовсім різні методи обчислень.

Текстовий плагіат у науковій сфері, на наш погляд, особливо шкідливий. В силу доступності наукової інформації виникає велика кількість прикладів яскраво вираженого плагіату на рівні наукових публікацій і кваліфікаційних робіт різного рівня, починаючи від рефератів і кваліфікаційних робота, закінчуючи докторськими дисертаціями. Псевдонауковці, що з'являються в результаті таких публікацій, негативно впливають на ефективність наукової спільноти в цілому.

Представлені тут міркування є переконливим свідченням актуальності завдання виявлення плагіату текстової інформації. З цим завданням досить успішно справляються експерти. Однак в цьому випадку існує ряд ускладнюючих факторів. По-перше, для успішного вирішення цього завдання експерт повинен мати здатність швидко зіставляти реферовані тексти з опублікованими раніше. Для цього експерт повинен або володіти великими знаннями в даній галузі, або вміти робити ефективний пошук необхідної

інформації за допомогою технічних засобів. Крім того, експерт повинен володіти розумінням предметної області, що дозволить йому виявляти плагіат не тільки на основі поверхневого зіставлення текстових шаблонів, але і на основі розуміння більш глибоких структурних і семантичних зв'язків.

На жаль, такий підхід до виявлення плагіату практично мало ефективний через масову потреби у виявленні плагіату і обмеженій кількості, і доступу до захищених експертам.

Автоматизація такої інтелектуальної діяльності експерта є складним завданням, яке повноцінно відноситься до категорії таких завдань штучного інтелекту як машинний переклад або навіть розуміння тексту на природній мові. Проте, актуальність цього завдання змушує дослідників і розробників постійно шукати шляхи все більш ефективного вирішення цього завдання.

Простий емпіричний досвід підказує, що найбільш ефективним способом створення тексту-плагіату є його практично прямий переклад на іншу мову. Такі випадки плагіату, як правило, не розпізнаються доступними сервісами автоматичного виявлення плагіату. У зв'язку з цим в контексті загальної задачі виявлення плагіату в даній роботі буде зроблений акцент на розробці автоматизованих засобів виявлення такого мультимовного виду плагіату. Таким чином, метою даної роботи є розробка і вдосконалення засобів виявлення мультимовного плагіату. З урахуванням зростаючої ролі української мови в різних сферах комунікації планується приділити особливу увагу параметрам функціонування подібних засобів з використанням української мови.

# ВИСНОВКИ

Розроблено та апробовано два методи визначення паралельних текстів:

- метод, заснований на аналізі розподілу частот слів відповідно до закону Зіпфа.

- метод, заснований на збігу високочастотних слів в документах.

Результати обчислювальних експериментів не дозволяють стверджувати, що будь-який з цих методів може бути використаний в якості надійного класифікатора.

Для підвищення якості класифікації запропонований метод обліку «висновків» кількох фільтрів з урахуванням їх класифікаційних метрик.

Спроектовано підхід до визначення парелельності текстів на основі векторного простору word2vec.

Перспективними надалі є роботи по розширенню корпусу лінгвістичних ресурсів, які можна застосувати в рамках запропонованого підходу.

# ЛІТЕРАТУРА


1. Hanks, P., Collins Dictionary of the English Language. // London: Collins, 1979. – 1690 p.
2. ЗАКОН УКРАЇНИ Про авторське право і суміжні права [Електронний ресурс] Режим доступу: http://zakon.rada.gov.ua/laws/show/3792-12
3. M. Kashkur, Research into Plagiarism Cases and Plagiarism Detection Methods / M. Kashkur, S. Parshutin, A. Borisov // Scientific Journal of Riga Technical University 2010
4. Обзор средств проверки уникальности контента [Електронний ресурс] Режим доступу: http://seomoney.org.ua/2010/04/obzor-sredstv-proverki-unikalnosti-kontenta
5. Handbook of Natural Language Processing. Second Edition // Nitin Indurkhya, Fred J. Damerau – published by Chapman & Hall/CRC, 2010. – 702p.
6. Marvin Minsky, Framework for Representing Knowledge // MIT-AI Laboratory, 1974. – 82 p.
7. John F. Sowa, Information Processing in Mind and Machine // Addison-Wesley, 1984. – 481 p.
8. INTERNATIONAL STANDARD ISO/IEC 24707 First edition 2007-10-11 Information technology — Common Logic (CL): a framework for a family of logic-based languages
9. D. Jurafsky Speech and Language Processing, 2nd Edition / D. Jurafsky, H. Martin // Prentice Hall 2008. – 1032 p.
10. Speech and Language Processing, 2nd Edition [Електронний ресурс] Режим доступа: https://github.com/rain1024/slp2-pdf
11. Mihalcea R., Graph-Based Natural Language Processing and Information Retrieval / Mihalcea R., Radev D. // Cambridge University Press 2011. – 202 p.

12. C. D. Manning, Foundations of Statistical Natural Language Processing / C. D. Manning, H. Schütze // The MIT Press 1999. – 620 p.
13. Steven Bird Natural Language Processing with Python / Steven Bird, Ewan Klein, Edward Loper // O'Reilly Media 2009. – 504 p.
14. Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit [Електронний ресурс] Режим доступу: https://www.nltk.org/book/
15. Benno Stein, Fuzzy-Fingerprints for Text-based Information Retrieval // Journal of Universal Computer Science. 2005. – P. 572-579.
16. Rashedur Rahman, Information Theoretical and Statistical Features for Intrinsic Plagiarism // SIGDIAL 2015 Conference. – P. 144-148.
17. Matt J. Kusner, From Word Embeddings To Document Distances / Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, Kilian Q. Weinberger // 32nd International Conference on Machine Learning. 2015. - P. 957–966.
18. Mikolov, T, Efficient estimation of word representations in vector space / Mikolov, T., Chen, K., Corrado, G., Dean, J. // In Proceedsings of Workshop at ICLR. 2013.
19. Jeffrey Pennington, GloVe: Global Vectors for Word Representation / Jeffrey Pennington, Richard Socher, Christopher D. Manning // Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014. – P. 1532 – 1543.
20. Vera Danilova, Cross-Language Plagiarism Detection Methods // Student Research Workshop associated with RANLP. 2013. - P.51-57
21. Steven T., Piantadosi Zipf’s word frequency law in natural language: A critical review and future directions // Psychon Bull Rev. 2014. - P. 1112–1130.
22. David M.W. Powers, Applications and Explanations of Zipfs Law. // New Methods in Language Processing and Computational Natural Language Learning, ACL – P. 151-160.

23. Орехова Ольга Михайловна, РЕАЛИЗАЦИЯ ЗАКОНА ЦИПФА НА МАТЕРИАЛЕ АНГЛИЙСКОГО И НЕМЕЦКОГО ЯЗЫКОВ // Грамота № 4. 2014. - С. 161-163.

24. Mehdi Mohammadi, Parallel Document Identification using Zipf’s Law // Ninth Workshop on Building and Using Comparable Corpora. 2016. - P. 21–25

25. Word2vec [Електронний ресурс] Режим доступу: https://code.google.com/archive/p/word2vec/

# Додаток А

## Реалізація Zipf-фільтра

### А.1 – Побудова і навчання Zipf-фільтра

```
import nltk, pylab
import math
import numpy
import scipy.optimize
import pickle

class MyLSM(object):
    def __init__(self, xdata, ydata):
        self.Xdata=xdata
        self.Ydata=ydata

    def SolveAB(self):
        rez= scipy.optimize.curve_fit(self.func, self.Xdata,
self.Ydata)
        return rez[0]

    def func(self, x, a, b):
        return a + b*x

class ZipfText2Area(object):
    def __init__(self, corpusFolder, fileName):
        self._ZipfNatural(corpusFolder, fileName)
        self._ZipfLogged()
        self._AB_LSM()

    def _ZipfNatural(self, corpusFolder, fileName):
        myreader =
nltk.corpus.PlaintextCorpusReader(corpusFolder,fileName+'.txt')
        print(fileName)
        fwords = myreader.words()
        fwords = self._Cleaning(fwords)
        freq_of_words = nltk.probability.FreqDist(fwords)
        freqList = [wordfreq for wordfreq in
freq_of_words.values()]

        sorted_and_ranked_freq_list =
tuple(enumerate(sorted(freqList, reverse = True),start = 1))
        sortedFreqList = sorted(list(map(lambda x: (x[1], x[0]),
freq_of_words.items())), reverse=True)

        ZipfDict =
tuple(zip(sorted_and_ranked_freq_list,[word[1] for word in
sortedFreqList]))
```

```
        list1 = [(i1,i2,i3) for ((i1,i2),i3) in ZipfDict]

        self.xRanks = [i1 for (i1,i2,i3) in list1]
        self.yFreqs = [i2 for (i1,i2,i3) in list1]
        self.freqProfile = [(list1[i][1],list1[i][2]) for i in
range(0, 200)]

    def _ZipfLogged(self):
        self.xLoggedRanks=[]
        for xRank in self.xRanks:
            self.xLoggedRanks.append(math.log10(xRank))
            self.yLoggedFreqs=[]
            for yFreq in self.yFreqs:
                self.yLoggedFreqs.append(math.log10(yFreq))

    def _AB_LSM(self):
        lsm = MyLSM(numpy.array(self.xLoggedRanks),
numpy.array(self.yLoggedFreqs))
        [self.a,self.b] = lsm.SolveAB()

    def _Cleaning(self, words):
        cleanWords=[]
        for w in words:
            if not (len(w)==1 and not w.isalpha()):
                cleanWords.append(w)
        return cleanWords

    def Area(self):
        return self.yLoggedFreqs[0]*self.xLoggedRanks[-1]/2.0

    def PlotNatural(self):
        pylab.plot(self.xRanks, self.yFreqs)
        pylab.show()

    def PlotLogged(self):
        pylab.plot(self.xLoggedRanks, self.yLoggedFreqs)
        pylab.show()

    def PlotLine(self):
        y1 = self.b*self.xLoggedRanks[0]+self.a
        y2 = self.b*self.xLoggedRanks[-1]+self.a
        pylab.plot([self.xLoggedRanks[0], self.xLoggedRanks[-
1]],[y1,y2])
        pylab.show()

class ZipfTextObjects(object):
    def __init__(self, corpusFolder, fileNames ):
        self.textObjects={}
        for fName in fileNames:
            fn=corpusFolder+'\\'+fName+'.pickle'
            print(fn)
            try:
```

```
                with open(fn, 'rb') as f:
                    zipfText2Area = pickle.load(f)
                print('Loading from pickle')
            except IOError as e:
                zipfText2Area =  ZipfText2Area(corpusFolder,
fName)
                with open(fn, 'wb') as f:
                    pickle.dump(zipfText2Area, f)
                print('Creating and saving to pickle')
            self.textObjects[fName] = zipfText2Area

class ZipfLinearLanguageRelation(object):
    def __init__(self, corpusFolder, fileNames1, fileNames2,
zipfTextObjects):
        self.scores1=self._TextScores(corpusFolder, fileNames1,
zipfTextObjects)
        self.scores2=self._TextScores(corpusFolder, fileNames2,
zipfTextObjects)
        self._AB_LSM()

    def _TextScores(self,corpusFolder,TextNames,
zipfTextObjects):
        scores = {}
        for name in TextNames:
            zipfText2Area =  zipfTextObjects.textObjects[name]
            scores[name]=zipfText2Area.Area()
        return scores

    def _AB_LSM(self):
        xList=list(self.scores1.values())
        yList=list(self.scores2.values())
        lsm = MyLSM(numpy.array(xList), numpy.array(yList))
        [self.a,self.b] = lsm.SolveAB()

    def Error(self, text1, text2, zipfTextObjects):
        xArea = zipfTextObjects.textObjects[text1]
        yArea = zipfTextObjects.textObjects[text2]
        # modelYScore = self.a+self.b*xArea
        modelYScore = self.a+self.b*xArea.Area()
        # return math.fabs(yArea-modelYScore)
        return math.fabs(yArea.Area() - modelYScore)

class ZipfDeltaLearning(object):
    def __init__(self):
        self.optimalDelta=-1

    def ParallelTextRecognition(self, text1, text2, delta,
zipfTextObjects):
        #area1 = zipfTextObjects.textObjects[text1]
        #area2 = zipfTextObjects.textObjects[text2]
        e=self.zipfLinearLanguageRelation.Error(text1, text2,
zipfTextObjects)
```

```
        return math.fabs(e)<=delta

    def BestParallelTextRecognition(self, text1, text2,
zipfTextObjects):
        return self.ParallelTextRecognition(text1, text2,
self.optimalDelta, zipfTextObjects)

    def _LearningExperiment(self, fileNames1, fileNames2, delta,
zipfTextObjects):
        #print(scores1)
        tp,tn,fp,fn = 0,0,0,0
        for i1 in range(0, len(fileNames1)):
            for i2 in range(0, len(fileNames2)):
#                print('pair: '+text1+' - '+text2)
                rez=self.ParallelTextRecognition(fileNames1[i1],
fileNames2[i2], delta, zipfTextObjects)
                if i1==i2: #texts really are parallel
                    if rez==True:
                        tp+=1
                    else:
                        fn+=1
                else:   #texts really are not  parallel
                    if rez==True:
                        fp+=1
                    else:
                        tn+=1
        return tp, tn, fp, fn

    def _Precision(self,tp,fp):
        if tp+fp==0:
            print('Presizion 0!')
            return 0
        return tp/(tp+fp)

    def _Recall(self,tp,fn):
        if tp+fn==0:
            print('Recall 0!')
            return 0
        return tp/(tp+fn)

    def _Accuracy(self, tp,tn,fn,fp):
        return (tp+tn)/(tp+tn+fp+fn)

    def _F1Metric(self, precision, recall):
        if precision+recall==0:
            print('F1 0!')
            return 0
        return 2*precision*recall/(precision+recall)

    def _ClassificationResults (self, fileNames1, fileNames2,
currentDelta, zipfTextObjects):
        #print('files1 - {}'.format(scores1))
```

```
        tp,tn,fp,fn = self._LearningExperiment(fileNames1,
fileNames2, currentDelta, zipfTextObjects)
        # print('tp={},tn={},fp={},fn={}'.format(tp,tn,fp,fn))
        precision = self._Precision(tp,fp)
        recall = self._Recall(tp,fn)
        f1 = self._F1Metric(precision, recall)
        # acr = self._Accuracy(tp,tn,fn,fp)
        return (tp,tn,fp,fn,precision,recall,f1)

    def SetThresholdRange(self, tresholdRange):
        self._tresholdRange =tresholdRange

    def FindOptimalAccuracyThreshold(self, corpusFolder,
fileNames1, fileNames2, zipfTextObjects):
        self.fileNames1=fileNames1
        self.fileNames2=fileNames2

self.zipfLinearLanguageRelation=ZipfLinearLanguageRelation(corpu
sFolder, fileNames1, fileNames2, zipfTextObjects)

        accuracyMax=-1
        deltaMax=-1

        for currentDelta in self._tresholdRange:
            [tp,tn,fp,fn,precision,recall,f1]=\
            self._ClassificationResults(fileNames1, fileNames2,
currentDelta, zipfTextObjects)
            accuracy = self._Accuracy(tp,tn,fp,fn)
            if accuracy > accuracyMax:
                accuracyMax = accuracy
                deltaMax=currentDelta
        self.optimalDelta=deltaMax
        return (deltaMax,accuracyMax)

    def FindOptimalF1Threshold(self, corpusFolder, fileNames1,
fileNames2, zipfTextObjects):
        self.fileNames1 = fileNames1
        self.fileNames2 = fileNames2
        self.zipfLinearLanguageRelation =
ZipfLinearLanguageRelation(corpusFolder, fileNames1, fileNames2,
zipfTextObjects)
        f1Max = -1
        deltaMax = -1
        precisionMax = -1
        recallMax = -1

        for currentDelta in self._tresholdRange:
            [tp, tn, fp, fn, precision, recall, f1] = \
                self._ClassificationResults(fileNames1,
fileNames2, currentDelta, zipfTextObjects)
            if f1 > f1Max:
                precisionMax = precision
```

```
                recallMax = recall
                f1Max = f1
                deltaMax = currentDelta
        self.optimalDelta = deltaMax
        return (deltaMax, precisionMax, recallMax, f1Max)

    def EvaluateTest(self, corpusFolder, fileNames1, fileNames2,
zipfTextObjects):
        #zipfLinearLanguageRelation =
ZipfLinearLanguageRelation(corpusFolder, fileNames1,
fileNames2,zipfTextObjects)
        scores1 =
self.zipfLinearLanguageRelation._TextScores(corpusFolder,
fileNames1, zipfTextObjects)
        scores2 =
self.zipfLinearLanguageRelation._TextScores(corpusFolder,
fileNames2, zipfTextObjects)
        (tp,tn,fp,fn,precision,recall,f1)=\
        self._ClassificationResults(fileNames1, fileNames2,
self.optimalDelta, zipfTextObjects)
        return (tp,tn,fp,fn,precision,recall,f1)
```

А.2 – Клієнтська частина Zipf-фільтра

```
RuTextNamesTrain = ['1r','2r','3r','4r','5r','6r','7r','8r']
UkTextNamesTrain = ['1u','2u','3u','4u','5u','6u','7u','8u']
RuTextNamesTest = ['9r','10r','11r','12r']
UkTextNamesTest = ['9u','10u','11u','12u']

zipfTextObjectsTrain = ZipfOOP.ZipfTextObjects(CorpusFolder,
RuTextNamesTrain+UkTextNamesTrain)
zipfTextObjectsTest = ZipfOOP.ZipfTextObjects(CorpusFolder,
RuTextNamesTest+UkTextNamesTest)

print('En vs Ru')
zipfDelta = ZipfOOP.ZipfDeltaLearning()

zipfLinearLanguageRelation =
ZipfOOP.ZipfLinearLanguageRelation(CorpusFolder,
RuTextNamesTrain,UkTextNamesTrain, zipfTextObjectsTrain)

for engNum in ['1','2','3','4','5','6','7','8']:
    for rusNum in ['1','2','3','4','5','6','7','8']:
        print(engNum + 'r' + ' - ' + rusNum + 'u' + ' : ',
end='')
        print(zipfLinearLanguageRelation.Error(engNum + 'r',
rusNum + 'u', zipfTextObjectsTrain))
    print()
zipfDelta.SetThresholdRange(arange(0.03,1.01,0.01))
print('deltaMax                precisionMax        recallMax
f1Max')
# print(zipfDelta.FindOptimalAccuracyThreshold(CorpusFolder,
```

```
RuTextNamesTrain, UkTextNamesTrain, zipfTextObjectsTrain))
print(zipfDelta.FindOptimalF1Threshold(CorpusFolder,
RuTextNamesTrain, UkTextNamesTrain, zipfTextObjectsTrain))

print (zipfDelta.EvaluateTest(CorpusFolder, RuTextNamesTest,
UkTextNamesTest, zipfTextObjectsTest))
```

Додаток Б

Реалізація фільтра на основі частотного розподілу слів

Б.1 – Побудова і навчання фільтра на основі частотного розподілу слів

```
from googletrans import Translator
from nltk.corpus import stopwords
from nltk.corpus import PlaintextCorpusReader
from nltk.probability import FreqDist
import pickle
from langdetect import detect

class WordFreqFilter(object):
    def __init__(self, corpusFolder, srcTexts, tgtTexts,
wordFreqCount):
        self.wordFreqCount = wordFreqCount
        #self.translateTo = translateTo
        self.srcPopWords = {}
        tgtPopWords = []

        for srcText in srcTexts:

self.srcPopWords[srcText]=self.__PopularWords(corpusFolder,
srcText)


        self.translatedPopWords = {}
        for tgtText in tgtTexts:
            #words = self.__PopularWords(corpusFolder, tgtText)
            fn = corpusFolder + '\\wfb' + tgtText + '.pickle'
            try:
                with open(fn, 'rb') as f:
                    self.translatedPopWords[tgtText] =
pickle.load(f)
                print('Loading from pickle')
            except IOError as e:
                words=self.__PopularWords(corpusFolder, tgtText)
                self.translatedPopWords[tgtText] =
self.__Translate(words)
                with open(fn, 'wb') as f:

pickle.dump(self.translatedPopWords[tgtText],f)
                print('Creating and saving to pickle')

    def __StrictCleaning(self, inFreqProfile):
        punkt = ['):--', ');', '),', ',', '.', ';', '-', "',-'",
"'", '"', '!', '--', '?', '."', '!"', '?"', '.,',
            ';--', '!--', ':', ', ', ',”', '".','!..', ').',
```

```
',"', '».', '»,','»','«',';&#','.].','.[.', '/', '〃',
            ']', '[', "?'", '(', ')', "!'", ",'", '"--', ',—',
'[...]', '.—', '…', '"[', '.]', '.,', '`', '-',
            '--"', r',"\'', '.)', "--'", ".'", ':--', r'?"\'',
r'"?\'', '...', '?〃', '.〃', '!〃', '",', '—', '&#',
            ',-', '.-', '!»', '?»', '…»', '//', '〃.', '~', '〃,',
'〃', '<', '>', '〃:', ',-', '.-']

        customStopwords = ['і', 'в', 'у', 'не'
,'що','він','на','я','з','з','як','а','то','все','вона','так','й
ого',
                    'але','так','ти','до','у','ж','ви','за' ,
'б', 'по', 'тільки', 'її', 'мені', 'було', 'ось', 'від',
                    'мене', 'ще', 'ні', 'про',
'з','йому','тепер','коли','навіть','ну','раптом','чи','якщо',
                    'вже','або','ні','бути' , 'був', 'нього',
'до', 'вас', 'небудь', 'знову', 'вже', 'ви', 'адже',
                    'там', 'потім', 'себе',
'нічого','їй','може','вони','тут','де','є',' треба
','ній','для',
                    'ми','тебе','їх' , 'ніж', 'була', 'сам',
'щоб', 'без', 'ніби', 'чого', 'раз', 'теж','собі', 'під',
                     'буде',
'ж','тоді','хто','цей','того','тому','цього','який','зовсім','ни
м','здесь','це','один','майже',

'мій','тим','щоб','неї','зараз','були','куди','навіщо','всіх' ,
'ніколи', 'можна', 'при', 'нарешті', 'два',
                    'про', 'інший', 'хоч', 'після', 'над',
'більше', 'той',
'через','ці','нас','про','все','них','яка','багато',
                    'хіба','три','цю','моя','втім' , 'добре',
'свою', 'цієї', 'перед', 'іноді', 'краще', 'трохи', 'тому', 'не
можна',
                    'такий', 'їм', 'більше',
'завжди','звичайно','всю','між'] + stopwords.words('russian')+
stopwords.words('english')+ punkt


        # outFreqProfile = [(f, w) for (f, w) in inFreqProfile
if not w in customStopwords and not w.isdigit()]
        outFreqProfile = [ w for w in inFreqProfile if not w in
customStopwords and not w.isdigit()]
        return outFreqProfile


    def __PopularWords(self, corpusFolder, fileName):
        myreader = PlaintextCorpusReader(corpusFolder, fileName
+ '.txt')
        fwords = myreader.words()
        fwordsLower=[]
        for w in fwords:
```

```
            fwordsLower.append(w.lower())
        cleanfwords = self.__StrictCleaning(fwordsLower)
        freq_of_words = FreqDist(cleanfwords)
        freqTupleList = [wordfreq for wordfreq in
freq_of_words.items()]
        sortedFreqList = sorted(list(map(lambda x: (x[1], x[0]),
freqTupleList)), reverse=True)
        sortedWords = [w for (f, w) in sortedFreqList]
        del sortedWords[self.wordFreqCount:]
        return sortedWords

        # goodWords=слова text без мусора (2)
        # goodWordsFreq=слова с частотами (FreqDist) (3)
        # отсортировать goodWordsFreq по убыванию частот
        # return popularWords= самые поп. из goodWordsFreq (без
частот)


    def __Translate(self, originalWordList):
        translatedWordList = []
        translatedWordListLowerCase = []
        translator = Translator()
        string1 = str(self.srcPopWords.values())
        string2 = str(originalWordList)
        # wordType = bool(re.search('[а-яА-Я]', string))
        # if wordType == True:
        #     translateTo = 'en'
        # else:
        #     translateTo = 'ru'
        srcWordLang = detect(string1)
        tgtWordLang = detect(string2)
        if srcWordLang == 'ru' and tgtWordLang == 'en':
            translateTo = 'ru'
        elif srcWordLang == 'uk' and tgtWordLang == 'en':
            translateTo = 'uk'
        elif srcWordLang == 'en' and tgtWordLang == 'ru':
            translateTo = 'en'
        elif srcWordLang == 'en' and tgtWordLang == 'uk':
            translateTo = 'en'
        elif srcWordLang == 'ru' and tgtWordLang == 'uk':
            translateTo = 'ru'
        elif srcWordLang == 'uk' and tgtWordLang == 'ru':
            translateTo = 'uk'
        else:
            raise Exception
        translations = translator.translate(originalWordList,
dest = translateTo)
        for translation in translations:
            translatedWordList.append(translation.text)
        for word in translatedWordList:
            translatedWordListLowerCase.append(word.lower())
        return translatedWordListLowerCase
```

```
    def IntersectionPercent(self, text1, text2):
        # counter - количество совпадений слов
        print('{0} - {1}'.format(text1, text2))
        for a in self.srcPopWords[text1]:
            print('{0:10}'.format(a), end=' ')
        print()
        for b in self.translatedPopWords[text2]:
            print('{0:10}'.format(b), end=' ')
        print()
        input('Next')
        counter = 0
        for a in self.srcPopWords[text1]:
            for b in self.translatedPopWords[text2]:
                if a == b:
                    counter+=1
                    #break
        percent = counter/self.wordFreqCount
        return percent

class WFBDeltaLearning(object):
    def __init__(self, wordFreqFilter):
        self.wordFreqFilter = wordFreqFilter
        self.optimalDelta = -1

    def ParallelTextRecognition(self, text1, text2, delta):
        #print("text1={0} text2={1}
percent={2}".format(text1,text2,self.wordFreqFilter.Intersection
Percent(text1, text2)))
        return self.wordFreqFilter.IntersectionPercent(text1,
text2) > delta

    def BestParallelTextRecognition(self, text1, text2):
        return self.ParallelTextRecognition(text1, text2,
self.optimalDelta)

    def _LearningExperiment(self, fileNames1, fileNames2,
delta):
        # print(scores1)
        tp, tn, fp, fn = 0, 0, 0, 0
        for i1 in range(0, len(fileNames1)):
            for i2 in range(0, len(fileNames2)):
                # print('pair: '+text1+' - '+text2)
                rez =
self.ParallelTextRecognition(fileNames1[i1], fileNames2[i2],
delta)
                if i1 == i2:
                    if rez == True:
                        tp += 1
                    else:
                        fn += 1
                else:
```

```
                    if rez == True:
                        fp += 1
                    else:
                        tn += 1
            print("=========================")
        return tp, tn, fp, fn

    def _Precision(self, tp, fp):
        if tp + fp == 0:
            #print('Presizion 0!')
            return 0
        return tp / (tp + fp)

    def _Recall(self, tp, fn):
        if tp + fn == 0:
            return 0
        return tp / (tp + fn)

    def _Accuracy(self, tp,tn,fn,fp):
        return (tp+tn)/(tp+tn+fp+fn)

    def _F1Metric(self, precision, recall):
        if precision + recall == 0:
            print('F1 0!')
            return 0
        return 2 * precision * recall / (precision + recall)

    def ClassificationResults(self, fileNames1, fileNames2,
currentDelta):
        # print('files1 - {}'.format(scores1))
        tp, tn, fp, fn = self._LearningExperiment(fileNames1,
fileNames2, currentDelta)
        precision = self._Precision(tp, fp)
        print("precision={0}".format(precision))
        recall = self._Recall(tp, fn)
        f1 = self._F1Metric(precision, recall)
        return (tp, tn, fp, fn, precision, recall, f1)

    def SetThresholdRange(self, tresholdRange):
        self._tresholdRange = tresholdRange

    def FindOptimalF1Threshold(self, fileNames1, fileNames2):
        self.fileNames1 = fileNames1
        self.fileNames2 = fileNames2
        f1Max = -1
        deltaMax = -1
        precisionMax = -1
        recallMax = -1

        for currentDelta in self._tresholdRange:
            [tp, tn, fp, fn, precision, recall, f1] = \
                self.ClassificationResults(fileNames1,
```

```
fileNames2, currentDelta)
            if f1 > f1Max:
                precisionMax = precision
                recallMax = recall
                f1Max = f1
                deltaMax = currentDelta
        self.optimalDelta = deltaMax
        return (deltaMax, precisionMax, recallMax, f1Max)

    def FindOptimalAccuracyThreshold(self, fileNames1,
fileNames2):
        self.fileNames1 = fileNames1
        self.fileNames2 = fileNames2
        AccuracyMax = -1
        deltaMax = -1

        for currentDelta in self._tresholdRange:
            [tp, tn, fp, fn, precision, recall, f1] = \
                self.ClassificationResults(fileNames1,
fileNames2, currentDelta)
            accuracy = self._Accuracy(tp, tn, fp, fn)
            if accuracy > AccuracyMax:
                AccuracyMax = accuracy
                deltaMax = currentDelta
        self.optimalDelta = deltaMax
        return (deltaMax, AccuracyMax)
```

Б.2 – Клієнтська частина фільтра на основі частотного розподілу слів

```
import WFBOOP
from numpy import arange

#CorpusFolder1 = r'C:\Users\Admin\Desktop\python\ZipfCorpus'
CorpusFolder2 = r'C:\Users\Admin\Desktop\python\Corpus100'

EnTextNamesTrain =
['1e','2e','3e','4e','5e','6e','7e','8e','9e','10e','11e','12e',
'13e','14e','15e','16e','17e','18e','19e','20e','21e']
RuTextNamesTrain =
['1r','2r','3r','4r','5r','6r','7r','8r','9r','10r','11r','12r',
'13r','14r','15r','16r','17r','18r','19r','20r','21r']
EnTextNamesTest =
['12e','13e','14e','15e','16e','17e','18e','19e','20e','21e']
RuTextNamesTest =
['12r','13r','14r','15r','16r','17r','18r','19r','20r','21r']


print('En vs Ru')
# WFBDeltaLearning
wordFreqFilter = WFBOOP.WordFreqFilter(CorpusFolder2,
EnTextNamesTrain, RuTextNamesTrain, 100)
```

```
wfbDeltaLearning = WFBOOP.WFBDeltaLearning(wordFreqFilter)

wfbDeltaLearning.SetThresholdRange(arange(0.05,0.3,0.01))

# learningResults =
wfbDeltaLearning.FindOptimalF1Threshold(EnTextNamesTrain,
RuTextNamesTrain)
learningResults =
wfbDeltaLearning.FindOptimalF1Threshold(RuTextNamesTrain,
EnTextNamesTrain)
print('////////////////////////')
print(learningResults)
print('////////////////////////')

# WFBDeltaTesting

wordFreqFilter = WFBOOP.WordFreqFilter(CorpusFolder2,
EnTextNamesTest, RuTextNamesTest, 50)
wfbDeltaTesting = WFBOOP.WFBDeltaLearning(wordFreqFilter)
a,b,c,d,e,f,F1=wfbDeltaTesting.ClassificationResults(EnTextNames
Test, RuTextNamesTest, learningResults[3])
print('tp={0}, tn={1}, fp={2}, fn={3}, precision={4},
recall={5}, F1={6}'.format(a,b,c,d,e,f,F1))
```

# Додаток В

## Код очищення тексту від зайвої інформації

```
from nltk.corpus import stopwords

def ItemsToWords(itemList):
    wordsLowerSplit = []
    for item in itemList:
        splitWords = item.split()
        for word in splitWords:
            wordsLowerSplit.append(word.lower())
    return wordsLowerSplit

def PunktCleaning(wordList):
    punkt = ['):--', ');', '),', ',', '.', ';', '-', "',-'",
"'", '"', '!', '--', '?', '."', '!"', '?"',
             ';--', '!--', ':', ', ', ',”', '".', '!..', ').',
',"', '».', '»,', '.,', '&#', '..', '...”',
             ']', '[', "?'", '(', ')', "!'", ",'", '"--', ',—',
'[...]', '.—', '--”', '------', '?...',
             '--"', r',"\'', '.)', "--'", ".'", ':--', r'?"\'',
r'"?\'', '...', '?”', '.”', '!”', '",',
             "'—", '/', '*', '**', ";'", '@', '[*]', '`']
    resultList = [w for w in wordList if not w in punkt and not
w.isdigit()]
    return resultList

def CleaningStopEn(wordList):
    stopW = stopwords.words('english') + ['the','The']
    resultList = [w for w in wordList if not w in stopW]
    return resultList

def CleaningStopRu(wordList):
    resultList = [w for w in wordList if not w in
stopwords.words('russian')]
    return resultList

def CleaningStopUk(wordList):
    stopWords = ['і', 'в', 'у', 'не', 'що', 'він', 'на', 'я',
'з', 'з', 'як', 'а', 'то', 'все', 'вона', 'так',
                 'його','так', 'ти', 'до', 'у', 'ж', 'ви',
'за', 'б', 'по', 'тільки', 'її', 'мені', 'було',
                 'але', 'ось', 'від','мене', 'ще', 'ні',
'про', 'з', 'йому', 'тепер', 'коли', 'навіть', 'ну', 'раптом',
'чи',
                 'якщо','вже', 'або', 'ні', 'бути', 'був',
'нього', 'до', 'вас', 'небудь', 'знову', 'вже', 'ви',
                 'адже','там', 'потім', 'себе', 'нічого',
```

```
'їй', 'може', 'вони', 'тут', 'де', 'є', ' треба ', 'ній',
                      'для','ми', 'тебе', 'їх', 'ніж', 'була',
'сам', 'щоб', 'без', 'ніби', 'чого', 'раз', 'теж', 'собі',
                      'під','буде', 'ж', 'тоді', 'хто', 'цей',
'того', 'тому', 'цього', 'який', 'зовсім', 'ним', 'здесь',
                      'це', 'один', 'майже','мій', 'тим',
'щоб', 'неї', 'зараз', 'були', 'куди', 'навіщо', 'всіх',
'ніколи', 'можна',
                      'при', 'нарешті', 'два','про', 'інший',
'хоч', 'після', 'над', 'більше', 'той', 'через', 'ці', 'нас',
'про', 'все',
                      'них', 'яка', 'багато','хіба', 'три',
'цю', 'моя', 'втім', 'добре', 'свою', 'цієї', 'перед', 'іноді',
'краще',
                      'трохи', 'тому', 'не можна','такий',
'їм', 'більше', 'завжди', 'звичайно', 'всю', 'між']

    resultList = [w for w in wordList if not w in stopWords]
    return resultList
```

# Додаток Г

## Єдиний метод навчання фільтрів

```
import math
class DeltaLearning(object):
    def __init__(self):
        self.optimalDelta = -1

    def SetThresholdRange(self, tresholdRange):
        self._tresholdRange = tresholdRange

    def FindOptimalAccuracyThreshold(self, trainSet,
errorEstimator):
        accuracyMax = -1
        deltaMax = -1
        for currentDelta in self._tresholdRange:
            [f1, accuracy] =
self._ClassificationResults(trainSet, currentDelta,
errorEstimator)
            if accuracy > accuracyMax:
                accuracyMax = accuracy
                deltaMax = currentDelta
        self.optimalDelta = deltaMax
        return (deltaMax, accuracyMax)

    def FindOptimalF1Threshold(self, trainSet, errorEstimator):
        f1Max = -1
        deltaMax = -1
        for currentDelta in self._tresholdRange:
            print("========== Current Delta {0}
====================".format(currentDelta))
            [f1, acr] = self._ClassificationResults(trainSet,
currentDelta, errorEstimator)
            if f1 > f1Max:
                f1Max = f1
                deltaMax = currentDelta
        self.optimalDelta = deltaMax
        return (deltaMax, f1Max)

    def _ClassificationResults(self, trainSet, currentDelta,
errorEstimator):
        tp, tn, fp, fn = self._LearningExperiment(trainSet,
currentDelta, errorEstimator)
        precision = self._Precision(tp, fp)
        recall = self._Recall(tp, fn)
        f1 = self._F1Metric(precision, recall)
        accuracy = self._Accuracy(tp, tn, fn, fp)
        return (f1, accuracy)
```

```
    def _LearningExperiment(self, trainSet, delta,
errorEstimator):
        tp, tn, fp, fn = 0, 0, 0, 0
        for trainItem in trainSet:
            text1 = trainItem[0][0]
            text2 = trainItem[0][1]
            fact = trainItem[1]
            rez = self.AreParallel(text1, text2, delta,
errorEstimator)
            if fact == True:  # texts really are parallel
                if rez == True:
                    tp += 1
                else:
                    fn += 1
            else:  # texts really are not  parallel
                if rez == True:
                    fp += 1
                else:
                    tn += 1
        return tp, tn, fp, fn

    def AreParallel(self, text1, text2, delta, errorEstimator):
        e = errorEstimator.TextSimilarity(text1, text2)
        return math.fabs(e) <= delta

    def _Precision(self, tp, fp):
        if tp + fp == 0:
            print('Presizion 0!')
            return 0
        return tp / (tp + fp)

    def _Recall(self, tp, fn):
        if tp + fn == 0:
            print('Recall 0!')
            return 0
        return tp / (tp + fn)

    def _Accuracy(self, tp, tn, fn, fp):
        return (tp + tn) / (tp + tn + fp + fn)

    def _F1Metric(self, precision, recall):
        if precision + recall == 0:
            print('F1 0!')
            return 0
        return 2 * precision * recall / (precision + recall)

    def BestParallelTextRecognition(self, text1, text2,
errorEstimator):
        return self.AreParallel(text1, text2, self.optimalDelta,
errorEstimator)

    def EvaluateTest(self, trainSet, errorEstimator):
```

```
        (f1, acr) = self._ClassificationResults(trainSet,
self.optimalDelta, errorEstimator)
        return (f1, acr)
```

Додаток Д

word2vec-фільтр

Д.1 – Реалізація системи на основі векторного уявлення word2vec (мономовний варіант)

```
import cleaningTools
from nltk.corpus import  PlaintextCorpusReader
from gensim.models import Word2Vec
from nltk.probability import FreqDist
from sklearn.metrics.pairwise import cosine_similarity
from math import fabs
import pickle


class ErrorEstimator(object):
    def __init__(self, texts, corpusFolder):
        self.repoFolder=corpusFolder
        file = corpusFolder + '\TextSimilarities.pickle'
        try:
            with open(file, 'rb') as f:
                self.similarityDict = pickle.load(f)
            print('Loading from pickle')
        except IOError:
            self.similarityDict = {}

        for text1 in texts:
            for text2 in texts:
                if (text1,text2) in self.similarityDict.keys():
                #try:
                    a= self.similarityDict[text1,text2]
                else:
                #except KeyError:
                    self.similarityDict[(text1, text2)] =
self.InnerTextSimilarity(text1, text2)

        with open(file, 'wb') as f:
            pickle.dump(self.similarityDict, f)

    def Cleaning(self, text):
        wordList1 = cleaningTools.ItemsToWords(text)
        wordList2 = cleaningTools.PunktCleaning(wordList1)
        wordList3 = cleaningTools.CleaningStopEn(wordList2)
        return wordList3

    def CommonWordsSearch(self, wordsEnClean1, wordsEnClean2):
```

```
wordCount=min(len(wordsEnClean1),len(wordsEnClean2),1000)
        freqDist1 = FreqDist(wordsEnClean1)
        popWordsFreq1 = freqDist1.most_common(wordCount)
        freqDist2 = FreqDist(wordsEnClean2)
        popWordsFreq2 = freqDist2.most_common(wordCount)
        tmp=[]
        for item1 in popWordsFreq1:
            for item2 in popWordsFreq2:
                if item1[0] == item2[0]:
                    rating=item1[1]+item2[1]- fabs(item1[1]-
item2[1])
                    tmp.append((rating,item1[0]))
        tmp=sorted(tmp, reverse=True)
        tmp=tmp[0:min(20,len(tmp))]
        rezult=[words for (rating, words) in tmp]
        #print (rezult)
        return rezult

    def Similarity(self, wordVectors, word1, word2):
        vector1 = wordVectors[word1][0]
        vector2 = wordVectors[word2][0]
        return cosine_similarity([vector1],[vector2])

    def Error(self, wordVectors1, wordVectors2, word1, word2):
        sim1 = self.Similarity(wordVectors1, word1, word2)
        sim2 = self.Similarity(wordVectors2, word1, word2)
        return fabs(sim1-sim2)

    def GetModelFromRepo(self, text):
        myreader = PlaintextCorpusReader(self.repoFolder, text)
        wordsEn = myreader.words(text)
        cleanText = self.Cleaning(wordsEn)
        fn = self.repoFolder + '\\' + text + '.w2vzipped'
        try:
            with open(fn, 'rb') as f:
                rezult = pickle.load(f)
            print('Loading from model ' + text)
        except IOError as e:
            model = Word2Vec([cleanText], size=100, window=3,
min_count=1, workers=4, sg=0)
            wordCount = []
            for word in model.wv.vocab:
                wordCount.append((model.wv.vocab[word].count,
word, model.wv[word]))
            wordCount = sorted(wordCount, reverse=True)
            wordCount = wordCount[:1000]
            rezult={}
            for item in wordCount:
                rezult[item[1]]=(item[2],item[0])
            with open(fn, 'wb') as f:
                pickle.dump(rezult, f)
            print('Creating and saving to model')
```

```
        return rezult

    def TotalError(self, wordVectors1, wordVectors2,
commonwords):
        ErrorSum = 0
        for word1 in commonwords:
            for word2 in commonwords:
                if word1 != word2:
                    ErrorSum+=self.Error(wordVectors1,
wordVectors2, word1, word2)
        return ErrorSum/(len(commonwords)*len(commonwords))

    def TextSimilarity(self, text1, text2):
        #try:
            return self.similarityDict[text1, text2]
       # except KeyError:
       #     print ("There is no calculated similarity for this
files")

    def InnerTextSimilarity(self, text1, text2):
        wordVectors1 = self.GetModelFromRepo(text1)
        wordVectors2 = self.GetModelFromRepo(text2)

        cleanText1 = wordVectors1.keys()
        cleanText2 = wordVectors2.keys()

        commonWords =
self.CommonWordsSearch(cleanText1,cleanText2)
        #print(commonWords)
        if commonWords==[]:
            return 1
        else:
            return self.TotalError(wordVectors1, wordVectors2,
commonWords)
```

Д.2 – Клієнтська частина word2vec-фільтра (мономовний варіант)

```
from Learning import DeltaLearning
from w2vErrorEstimatorMono import ErrorEstimator
from numpy import arange


def PrepareTtrainSet(TrainTextNames):
    trainSet = []
    for index1 in range(0, len(TrainTextNames)):
        for index2 in range(index1, len(TrainTextNames)):
            if index1 == index2:
                trainSet.append([(TrainTextNames[index1],
TrainTextNames[index2]), 1])
            else:
                trainSet.append([(TrainTextNames[index1],
```

```
TrainTextNames[index2]), 0])
    return trainSet

def SetParallel(text1, text2, trainSet):
    for item in trainSet:
        if item[0][0]==text1 and item[0][1]==text2:
            item[1] = 1


CorpusFolder=r'E:\py projects\python\vectorCorpus'

TrainTextNames = ['1e.txt',
'1eFake.txt','2e.txt','2eFake.txt','3e.txt','3eFake.txt','4e.txt
','4eFake.txt',
                  '5e.txt', '5eFake.txt',
'6e.txt','6eFake.txt','7e.txt', '7eFake.txt',

'8e.txt','9e.txt','9eFake.txt','10e.txt','10eFake.txt']
trainSet = PrepareTtrainSet(TrainTextNames)
SetParallel("1e.txt", "1eFake.txt",trainSet)
SetParallel("2e.txt", "2eFake.txt",trainSet)
SetParallel("3e.txt", "3eFake.txt",trainSet)
SetParallel("4e.txt", "4eFake.txt",trainSet)
SetParallel("5e.txt", "5eFake.txt",trainSet)
SetParallel("6e.txt", "6eFake.txt",trainSet)
SetParallel("7e.txt", "7eFake.txt",trainSet)
SetParallel("9e.txt", "9eFake.txt",trainSet)
SetParallel("10e.txt", "10eFake.txt",trainSet)
print(trainSet)

errorEstimator=ErrorEstimator(TrainTextNames, CorpusFolder)


deltaLearning = DeltaLearning()
#deltaLearning.SetThresholdRange(arange(0.01,1.01,0.01))
deltaLearning.SetThresholdRange(arange(0.0001,0.010,0.0001))

deltaMax,F1 = deltaLearning.FindOptimalF1Threshold(trainSet,
errorEstimator)
print("deltaMax={0} F1={1}".format(deltaMax,F1))

TestTextNames = [ '11e.txt', '12e.txt', '13e.txt', '14e.txt',
'14eFake.txt', '15e.txt', '15eFake.txt',
                  '16e.txt', '17e.txt', '17eFake.txt',
'18e.txt', '18eFake.txt',
                  '19e.txt', '20e.txt', '20eFake.txt','21e.txt',
'21eFake.txt']
testSet = PrepareTtrainSet(TestTextNames)
SetParallel("14e.txt", "14eFake.txt",testSet)
SetParallel("15e.txt", "15eFake.txt",testSet)
SetParallel("17e.txt", "17eFake.txt",testSet)
SetParallel("18e.txt", "18eFake.txt",testSet)
```

```
SetParallel("20e.txt", "20eFake.txt",testSet)
SetParallel("21e.txt", "21eFake.txt",testSet)
print(testSet)

errorEstimator=ErrorEstimator(TestTextNames, CorpusFolder)

F2,acr=deltaLearning.EvaluateTest(testSet, errorEstimator)
print("F1={0} Accuracy={1}".format(F2, acr))
```