ДОДАТОК А

Програмний код

```
public class Word2VecCalculator : IVocabularyVectorCalculator
    {
        private readonly Vocabulary _vocabulary;
        private readonly ITokenizer _tokenizer;

        public Word2VecCalculator(IWord2VecTextReader word2VecTe
xtReader, IVectorProvider _vectorProvider, ITokenizer tokenizer
)
        {
            _tokenizer = tokenizer;
            var vectors = _vectorProvider.GetVectorDataset();
            _vocabulary = word2VecTextReader.Read(vectors);
        }

        public Representation GetRepresentationForTerm(string te
rm)
        {
            return _vocabulary.GetRepresentationOrNullFor(term);
        }

        public Representation GetRepresentationForDoc(string doc
ument)
        {
            var tokens = _tokenizer.Tokenize(document);

            Representation representation = null;
            foreach (var item in tokens)
            {
```

```
                var tokenRepresentation = _vocabulary.GetRepresentationOrNullFor(item);
                if (tokenRepresentation == null)
                {
                    continue;
                }

                if (representation == null)
                {
                    representation = tokenRepresentation;
                }
                else
                {
                    representation.Add(tokenRepresentation);
                }
            }

            return representation;
        }
    }

public class TrainDataLoader : ITrainDataLoader
    {
        private readonly IVocabularyVectorCalculator _vectorCalculator;
        private readonly IFileReader _fileReader;
        private readonly ISerializer _serializer;

        public TrainDataLoad(IVocabularyVectorCalculator vectorCalculator, IFileReader fileReader, ISerializer serializer)
        {
            _vectorCalculator = vectorCalculator;
            _fileReader = fileReader;
```

```
            _serializer = serializer;
        }

        public async Task<TrainDataModel> LoadAsync(string train
DataPath)
        {
            var newsDataSet = await _fileReader.ReadFileAsync(tr
ainDataPath);
            var news = _serializer.Deserialize<IEnumerable<NewsM
odel>>(newsDataSet);

            var newsVectors = ConvertToVectors(news);

            return newsVectors;
        }


        private TrainDataModel ConvertToVectors(IEnumerable<News
Model> news)
        {
            var categories = new List<int>();
            var newsVectors = new List<double[]>();

            foreach (var group in news)
            {
                if (!Enum.TryParse(typeof(Categories), group.Cat
egory, true, out var category))
                {
                    continue;
                }

                var vector = _vectorCalculator.GetRepresentation
ForDoc(group.Text)?.NumericVector.ToDouble();
```

```
                if (vector == null || vector.Length == 0)
                {
                    continue;
                }

                newsVectors.Add(vector);
                categories.Add((int)category);
            }

            return new TrainDataModel(newsVectors, categories);
        }
    }


public class SvgClassifier : ITextClassifier
  {
      private MulticlassSupportVectorMachine<Linear> _context;

      public void TrainModel(TrainDataModel model)
      {
          if (model.Vectors.Length != model.Categories.Length)
          {
              throw new InvalidOperationException("Vectors amou
nt and Categories amount should be the same.");
          }

          var teacher = new MulticlassSupportVectorLearning<Lin
ear>
          {
              Learner = p => new LinearDualCoordinateDescent
              {
                  Loss = Loss.L2
              }
```

```
            };

            _context = teacher.Learn(model.Vectors, model.Categor
ies);
        }

        public Categories Predict(double[] documentVector)
        {
            var teacher = new MulticlassSupportVectorLearning<Lin
ear>
            {
                Learner = p => new LinearDualCoordinateDescent
                {
                    Loss = Loss.L2
                }
            };

            var predictedValue = _context.Decide(documentVector);
            var category = Enum.Parse(typeof(Categories), predict
edValue.ToString());
            return category;
        }
    }


    public class CrossTest : IClassifierTest
    {
        private readonly ITextClassifier _classifier;

        public CrossTest(ITextClassifier classifier)
        {
            _classifier = classifier;
        }
```

```
        public int TestClassifier(TrainDataModel model)
        {
            var vectorsLength = model.Vectors.Length;
            var categoriesLength = model.Categories.Length;
            if (vectorsLength != categoriesLength)
            {
                throw new InvalidOperationException("Vectors numb
er and Categories number should be the same.");
            }

            var successCount = 0;
            for (int documentIndex = 0; documentIndex < vectorsLe
ngth; documentIndex++)
            {
                var predictedValue = _classifier.Predict(model.Ve
ctors[documentIndex]);
                if (predictedValue == categoriesLength[documentIn
dex])
                {
                    successCount++;
                }
            }

            return successCount * vectorsLength / 100;
        }
    }

public class Tokenizer : ITokenizer
    {
        public IEnumerable<string> Tokenize(string input)
        {
            var tokens = new List<string>();
```

```
            var sb = new StringBuilder();

            var arr = input.ToCharArray();
            for (var i = 0; i < arr.Length; i++)
            {

                var prior = (i - 1 > 0) ? arr[i - 1] : ' ';
                var current = arr[i];
                var next = (i + 1 < arr.Length) ? arr[i + 1] : '
';

                if (char.IsLetter(current) && '.' == next)
                {

                    if (i + 5 < input.Length)
                    {

                        if (char.IsLetter(arr[i])
                            && '.' == arr[i + 1]
                            && char.IsLetter(arr[i + 2])
                            && '.' == arr[i + 3]
                            && char.IsLetter(arr[i + 4]))
                        {

                            for (; i < arr.Length && arr[i] != '
'; i++)
                            {
                                sb.Append(arr[i]);
                            }

                            if (i + 1 < input.Length
                                && '.' == arr[i + 1])
                            {
```

```
                    sb.Append(arr[i++]);
                }

                AddToken(tokens, sb.ToString());
                sb = new StringBuilder();
                continue;
            }
        }
    }

    if ('w' == current && '/' == next)
    {
        sb.Append(current);
        sb.Append(next);
        AddToken(tokens, sb.ToString());
        sb = new StringBuilder();
        i += 1;
        continue;
    }

    if ('h' == current && 't' == next)
    {
        if (i + 7 < input.Length &&
            "http://".Equals(input.Substring(i, i +
7)))
        {

            for (; i < arr.Length && arr[i] != ' ';
i++)
            {
                sb.Append(arr[i]);
            }
```

```
                AddToken(tokens, sb.ToString());
                sb = new StringBuilder();
                continue;
            }
        }

        if (char.IsLetter(current) && ':' == next)
        {
            if (i + 2 < input.Length
                && (arr[i + 2] == '\\'
                    || arr[i + 2] == '/'))
            {
                sb.Append(current);
                sb.Append(next);
                sb.Append(arr[i + 2]);
                i += 2;
                continue;
            }
        }

        if (char.IsDigit(current) && '.' == next)
        {
            if (i + 2 < input.Length &&
                char.IsDigit(arr[i + 2]))
            {
                sb.Append(current);
                sb.Append(next);
                sb.Append(arr[i + 2]);
                i += 2;
                continue;
            }
        }
```

```
if (char.IsLetter(current) && '-' == next)
{
    if (i + 2 < input.Length &&
        char.IsLetter(arr[i + 2]))
    {
        sb.Append(current);
        sb.Append(next);
        sb.Append(arr[i + 2]);
        i += 2;
        continue;
    }
}

if (char.IsLetter(current) && '_' == next)
{
    if (i + 2 < input.Length &&
        char.IsLetter(arr[i + 2]))
    {
        sb.Append(current);
        sb.Append(next);
        i++;
        continue;
    }
}

if (('#' == current ||
     '@' == current) &&
    ' ' != next &&
    !char.IsSymbol(next))
{
    sb.Append(current);
    continue;
}
```

```
if (' ' != current && '/' == next)
{
    sb.Append(current);
    sb.Append(next);
    i++;
    continue;
}
if (' ' == current)
{
    AddToken(tokens, sb.ToString());
    sb = new StringBuilder();
    continue;
}
if ('\'' == current)
{
    if (' ' == prior)
    {
        AddToken(tokens, "'");
    }
    else
    {
        sb.Append(current);
    }
    continue;
}
if (char.IsSymbol(current))
{
    AddToken(tokens, sb.ToString());
    AddToken(tokens, current.ToString());
    sb = new StringBuilder();
    continue;
}
```

```
                sb.Append(current);
            }
            if (0 != sb.Length)
            {
                AddToken(tokens, sb.ToString());
            }
            return tokens;
        }
        private void AddToken(List<string> tokens, string value)
        {
            tokens.Add(value);
        }
    }
public class Normalizer : INormalizer
    {
        private readonly IUaDictionaryRepository _uaDictionary;
        private readonly IEnDictionaryRepository _enDictionary;

        public Normalizer(IUaDictionaryRepository uaDictionary,
IEnDictionaryRepository enDictionary)
        {
            _uaDictionary = uaDictionary;
            _enDictionary = enDictionary;
        }
        public async Task<IEnumerable<string>> NormalizeAsync(Li
st<string> input)
        {
            var succededCount = 0;
            var inputCount = input.Count;
            const int attemptCountDefault = 7;
            var attemptCount = inputCount < attemptCountDefault
? inputCount : attemptCountDefault;
            var lemmatized = new List<string>(input.Count);
```

```
            for (var i = 0; i < attemptCount; i++)
            {
                var lemma = await _enDictionary.GetLemmaAsync(in
put.ElementAtOrDefault(i));
                if (lemma != null)
                {
                    succededCount++;
                    lemmatized.Add(lemma.Lemma);
                }
            }

            if (succededCount >= 4)
            {
                for (var i = attemptCount; i < inputCount; i++)
                {
                    var lemma = await _enDictionary.GetLemmaAsyn
c(input.ElementAtOrDefault(i));
                    if (lemma != null)
                    {
                        succededCount++;
                        lemmatized.Add(lemma);
                    }
                }

                return lemmatized;
            }

    lemmatized = new List<string>(input.Count);
        foreach (var token in input)
            {
                var enLemma = await _uaDictionary.GetLemmaAsync(
token);
                if (enLemma != null)
```

```
                {
                    lemmatized.Add(enLemma.Lemma);
                }
            }

            return lemmatized;
        }
    }
public static class ModelBuilder
    {
        private static string TRAIN_DATA_FILEPATH = @"C:\Users\O
ksana_Serdiuk\Downloads\news-category-
dataset\News_Category_Dataset_v2.csv";
        private static string MODEL_FILEPATH = @"../../../../Tex
tMiningML.Model/MLModel.zip";

        private static MLContext mlContext = new MLContext(seed:
 1);

        public static void CreateModel()
        {
            var trainingDataView = mlContext.Data.LoadFromTextFi
le<ModelInput>(
                                                path: TRAIN_DATA_FIL
EPATH,
                                                hasHeader: true,
                                                separatorChar: ',',
                                                allowQuoting: true,
                                                allowSparse: false);

            var trainingPipeline = BuildTrainingPipeline(mlConte
xt);
```

```
            var mlModel = TrainModel(mlContext, trainingDataView
, trainingPipeline);

            SaveModel(mlContext, mlModel, MODEL_FILEPATH, traini
ngDataView.Schema);
        }

        public static IEstimator<ITransformer> BuildTrainingPipe
line(MLContext mlContext)
        {
            var dataProcessPipeline = mlContext.Transforms.Conve
rsion.MapValueToKey("Category", "Category")
                                        .Append(mlContext.Transfor
ms.Text.FeaturizeText("Title_tf", "Title"))
                                        .Append(mlContext.Transfor
ms.Text.FeaturizeText("Description_tf", "Description"))
                                        .Append(mlContext.Transfor
ms.Concatenate("Features", new[] { "Title_tf", "Description_tf"
 }))
                                        .Append(mlContext.Transfor
ms.NormalizeMinMax("Features", "Features"))
                                        .AppendCacheCheckpoint(mlC
ontext);

            var trainer = mlContext.MulticlassClassification.Tra
iners.OneVersusAll(mlContext.BinaryClassification.Trainers.Aver
agedPerceptron(labelColumnName: "Category", numberOfIterations:
 10, featureColumnName: "Features"), labelColumnName: "Category
")
                                        .Append(mlContext.Transfor
ms.Conversion.MapKeyToValue("PredictedLabel", "PredictedLabel")
);
```

```
            var trainingPipeline = dataProcessPipeline.Append(tr
ainer);

            return trainingPipeline;
        }

        public static ITransformer TrainModel(MLContext mlContex
t, IDataView trainingDataView, IEstimator<ITransformer> trainin
gPipeline)
        {
            var model = trainingPipeline.Fit(trainingDataView);
            return model;
        }

        private static void SaveModel(MLContext mlContext, ITran
sformer mlModel, string modelRelativePath, DataViewSchema model
InputSchema)
        {
            mlContext.Model.Save(mlModel, modelInputSchema, GetA
bsolutePath(modelRelativePath));
        }

        public static string GetAbsolutePath(string relativePath
)
        {
            var _dataRoot = new FileInfo(typeof(Program).Assembl
y.Location);
            var assemblyFolderPath = _dataRoot.Directory.FullNam
e;

            var fullPath = Path.Combine(assemblyFolderPath, rela
tivePath);
```

```
            return fullPath;
        }

        public static double CalculateStandardDeviation(IEnumera
ble<double> values)
        {
            var average = values.Average();
            var sumOfSquaresOfDifferences = values.Select(val =>
 (val - average) * (val - average)).Sum();
            var standardDeviation = Math.Sqrt(sumOfSquaresOfDiff
erences / (values.Count() - 1));
            return standardDeviation;
        }
    }
```

ДОДАТОК Б

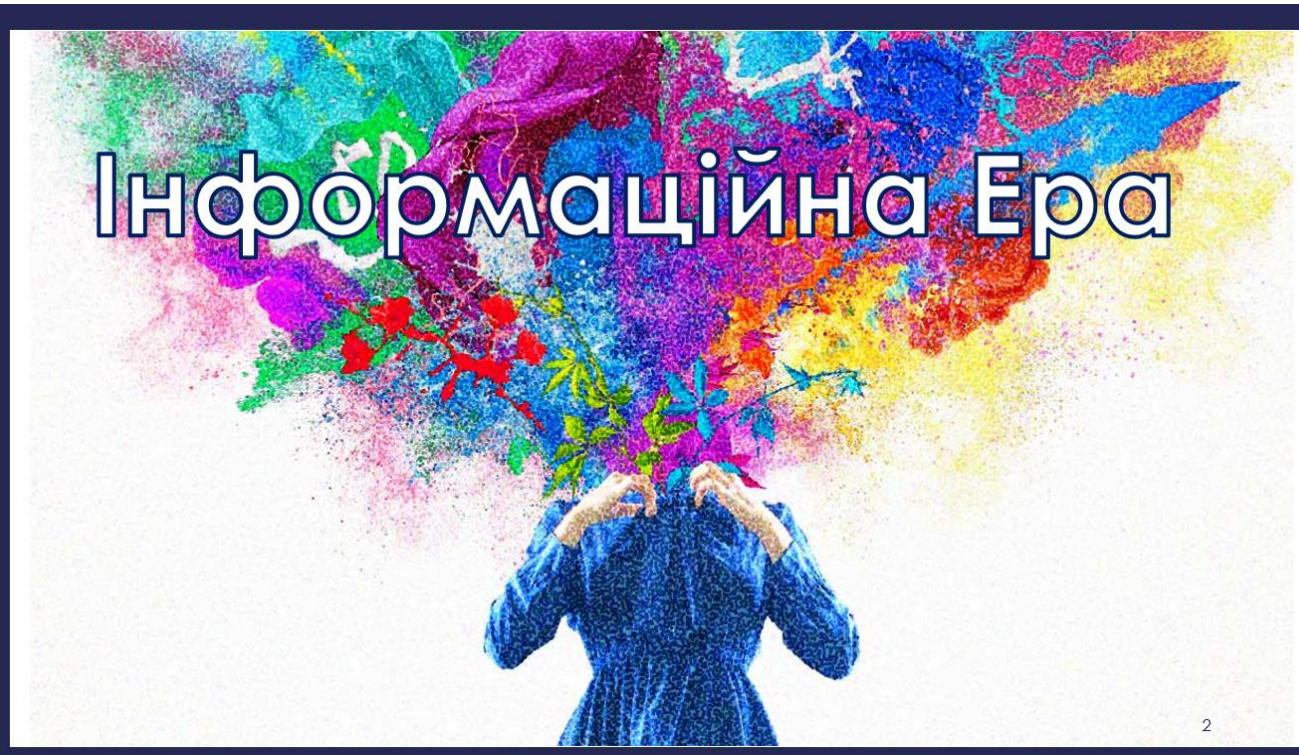
Слайди презентації

АТЕСТАЦІЙНА РОБОТА МАГІСТРА

ДОСЛІДЖЕННЯ МЕТОДІВ ТА ПРОГРАМНИХ ЗАСОБІВ АНАЛІЗУ НОВИННИХ СТРІЧОК В СОЦІАЛЬНИХ МЕРЕЖАХ

Виконала:
ст. гр. ІПЗМ-17-2
Сердюк О.О.

Керівник:
проф. к.т.н. Єрохін А.Л.



Джерела інформації:
- соціальні мережі;
- новини, публікації;
- активність користувачів в Інтернеті;
- активність людей поза межами Інтернету;
- активність веб-додатків.

Проблема: сегментація, класифікація даних, статистика і передбачення на її основі.
Рішення: Data Mining.

3

Мета роботи

Дослідження існуючих методів аналізу текстів взятих з новинних стрічок соціальних мереж, визначення категорії тексту.

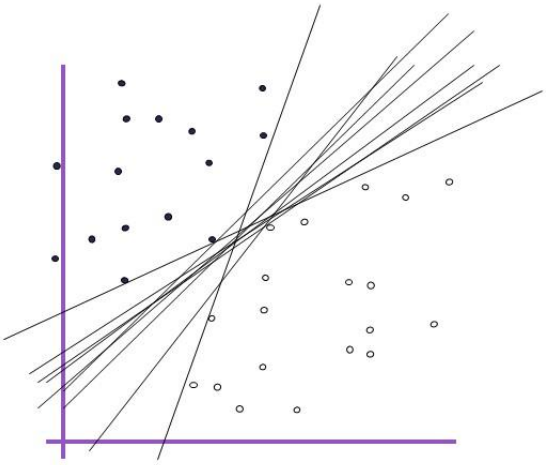
Написання веб-додатку, що дозволить користувачеві розділити свою новинну стрічку з соціальної мережі Facebook на категорії.

4

Класифікація

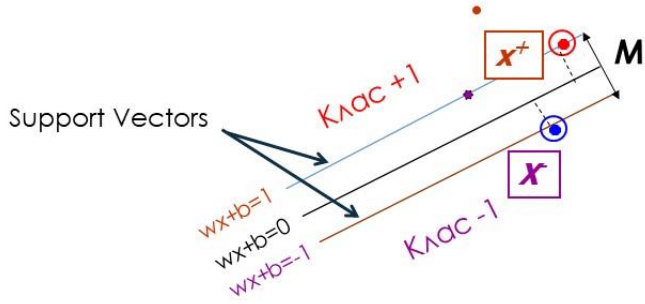
5

Лінійна класифікація



6

Лінійна класифікація



$$M = \frac{(x^+ - x^-) \cdot w}{|w|} = \frac{2}{|w|}$$

7

Мультикласова класифікація

- Умова: кожен елемент належить саме одному класові.

Приклад

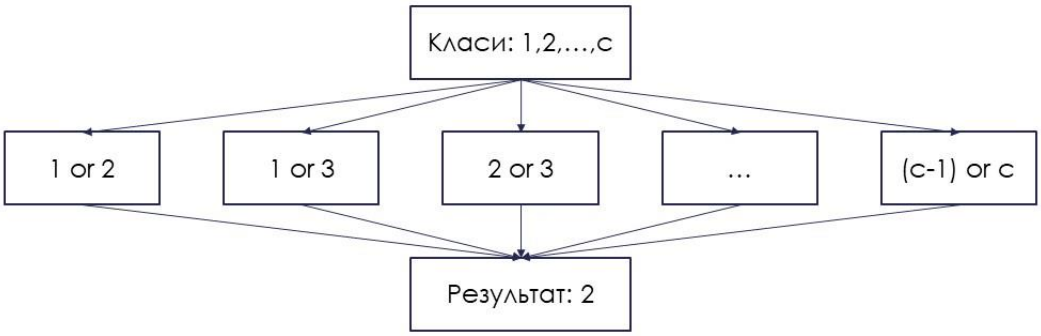
- Вхідні дані: «У країні відбулися вибори президента країни.»
- Результат: політика.

- Вхідні дані: «У прокат вийшла остання кінострічка «Месники».»
- Результат: кіно.

8

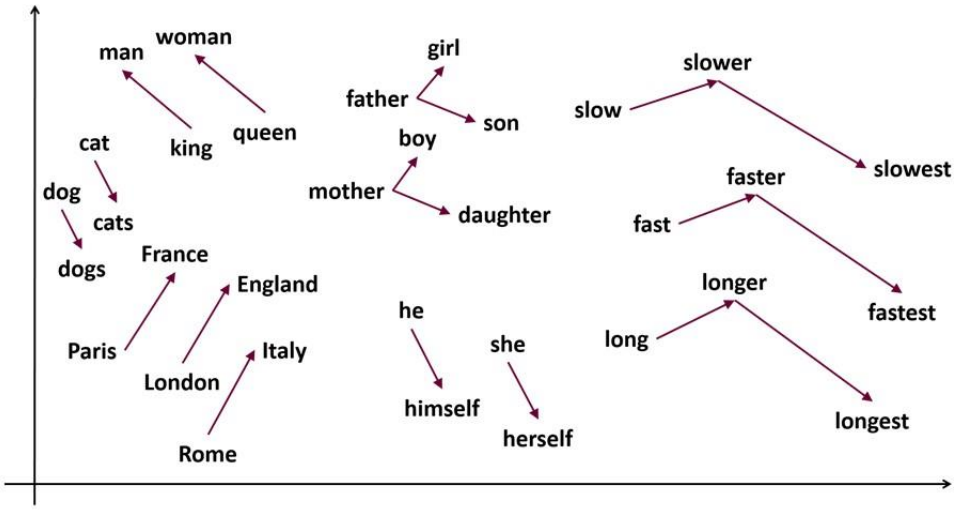
SVM (1-vs-1)

- Використовує k(k-1)/2 бінарних класифікатори, результати яких агреговані в фінальне рішення.



9

Word embedding - Word2Vec



10

ML.NET

- ✓ Фреймворк для машинного навчання на платформі .NET.
- ✓ Підтримка стеку Microsoft (Azure, Power BI).
- ✓ Крос-платформність.
- ✓ Дата релізу стабільної версії: 4 червня 2019

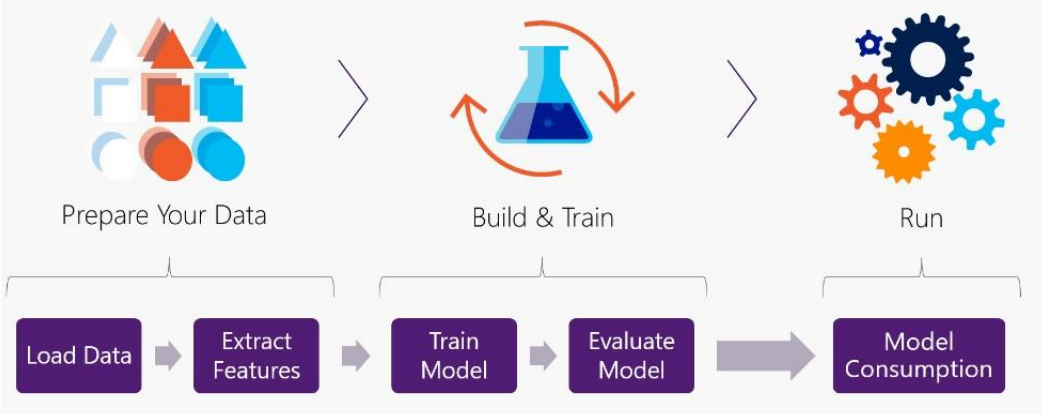
Задачі:
- Binary Classification
- Multi-Class classification
- Regression
- Ranking
- Anomaly Detection
- Clustering
- Recommendation (preview)

ML.NET

11

ML.NET

Використаний алгоритм: метод бінарної класифікації Averaged Perceptron (1 vs All).



12

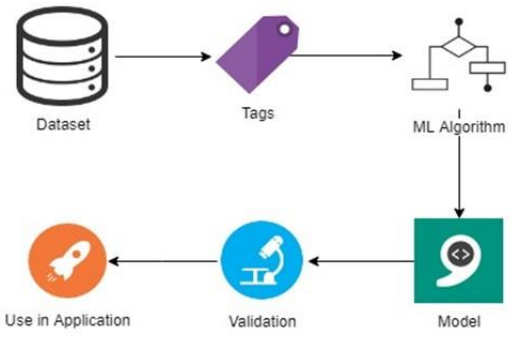
Постановка задачі

Розробити програмний продукт, що дозволить авторизуватися за допомогою соціальної мережі Facebook та, надавши доступ до своєї сторінки, переглянути свою новинну стрічку розбиту на категорії.

Порівняти методи аналізу тексту реалізованого за допомогою SVM та за допомогою ML.

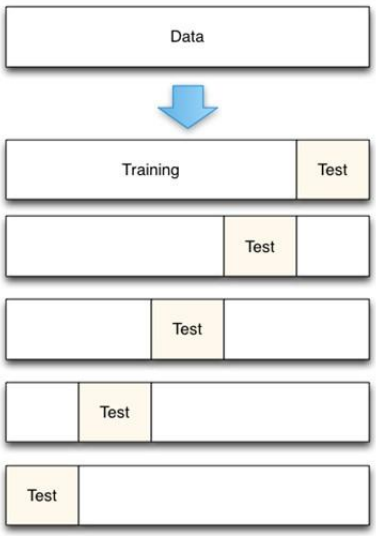
13

Етапи реалізації програмної системи



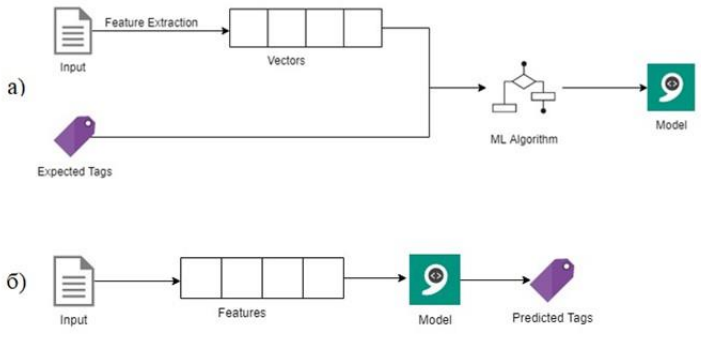
14

Перехресна перевірка



15

Тренування і використання моделі класифікації



16

Технології

Microsoft Azure

ML.NET

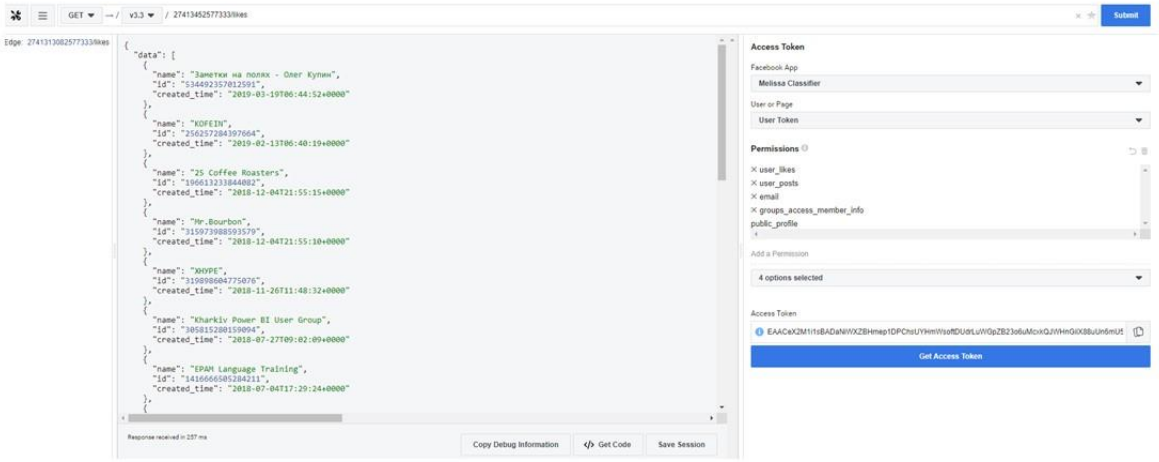
facebook

OAUTH 2

ASP.NET Core

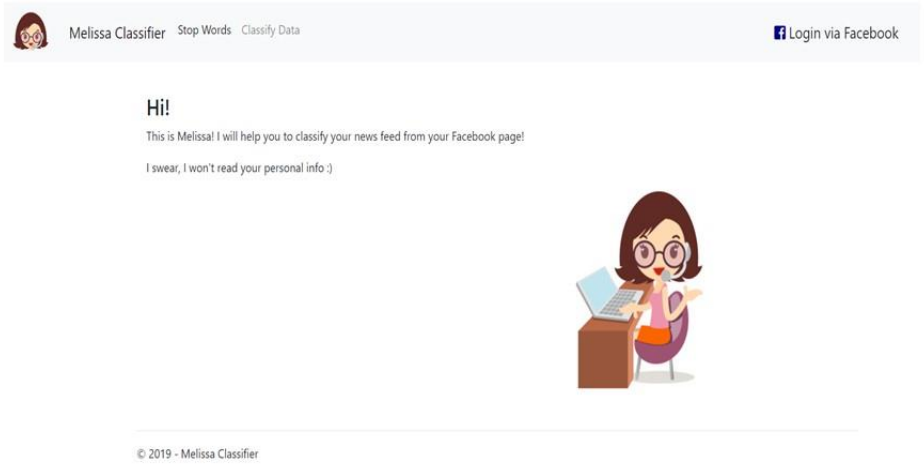
17

Приклад запиту до Facebook Graph API



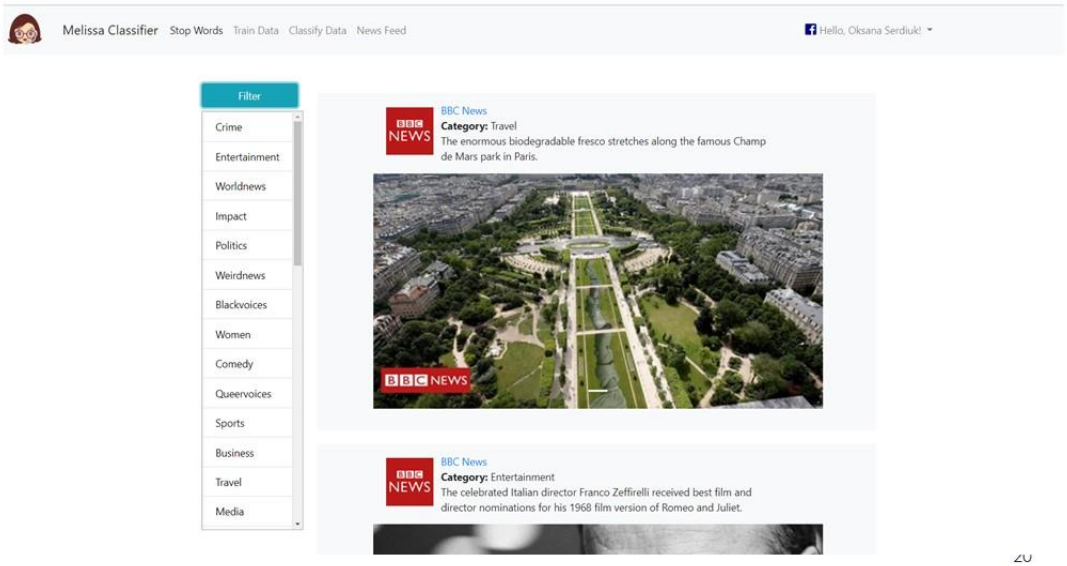
18

Головна сторінка продукту



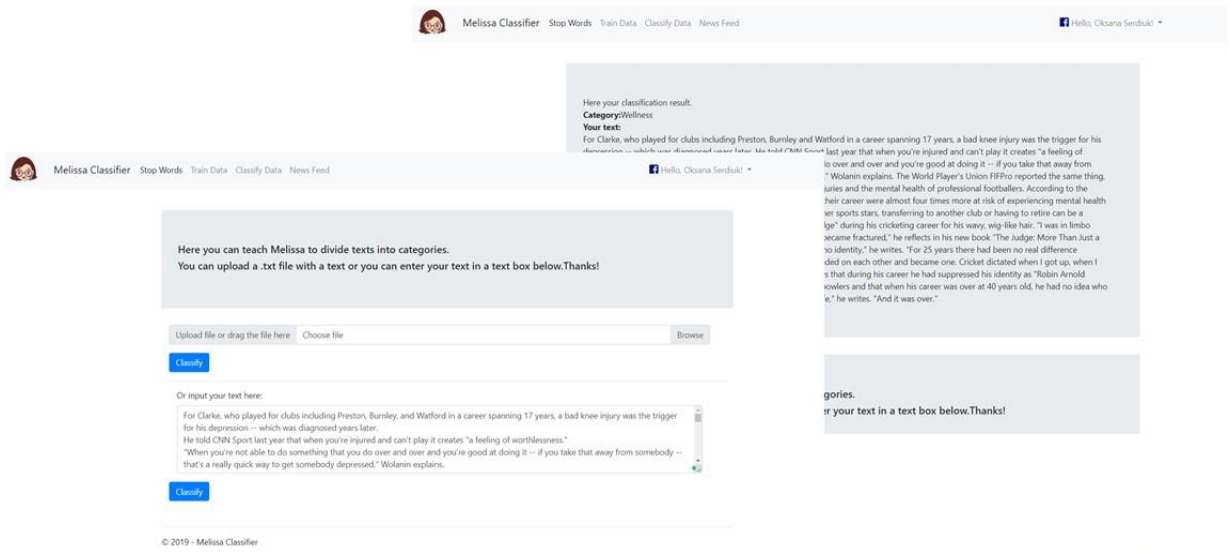
19

Сторінка з новинною стрічкою



20

Сторінка з новинною стрічкою



21

Висновки

Було розроблено веб-додаток для класифікації текстів, що дозволяє класифікувати за допомогою двох алгоритмів: SVM у поєднанні з Word2Vec та ML.NET.

У результаті порівняння виявлено:
SVM потребує більше вхідних даних для навчання.

ML.NET пропонує швидкий спосіб вирішення багатьох задач машинного навчання. Швидкий і ефективний у роботі.

За умови якісного навчання SVM може зрівнятися з ML.NET у ефективності.

22

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Міжнародна весняна школа
з верифікації та штучного інтелекту

СТЕНДОВІ ДОПОВІДІ

м. Харків, 24-25 квітня 2018 р.

Зміст

1. ДОСЛІДЖЕННЯ МЕТОДІВ ТА ПРОГРАМНИХ ЗАСОБІВ АНАЛІЗУ НОВИННИХ СТРІЧОК В СОЦІАЛЬНИХ МЕРЕЖАХ. Сердюк О.О., студент магістратури ПІ. Науковий керівник: д.т.н., проф. Єрохін А.Л. 3
2. CODE QUALITY ANALYSIS SYSTEM BASED ON GENERAL UML DIAGRAMS WITH SUPPORT OF MACHINE LEARNING. Ostapenko D.S., graduate student SE Department. Academic adviser: D. Sc., Professor Andriy L. Yerokhin.....10
3. АВТОМАТИЗОВАНА СИСТЕМА ОБЛІКУ ВІДВІДУВАНЬ ЗАНЯТЬ І ГЕНЕРАЦІЇ ЗВІТІВ. Стьопін В.І., студент кафедри СТ, Горбатенко Б.В., студент кафедри СТ. Науковий керівник: к.т.н., проф. Іванов В.Г.....15
4. ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ОЦІНЮВАННЯ ТЕПЛОМАСООБМІННИХ ПРОЦЕСІВ У МІКРОАНАЛАХ. Зацеркляний Г.А., аспірант кафедри ПІ ХНУРЕ. Науковий керівник: д.т.н., проф. Єрохін А.Л.....22
5. ANALYSIS OF CHATBOT VERIFICATION METHODS IN MESSENGERS. Artur Vasyliev, graduate student SE Department. Academic adviser: PhD, Prof. I.Shubin.....32
6. ДОСЛІДЖЕННЯ МОДЕЛЕЙ ОЦІНЮВАННЯ ТОНАЛЬНОГО ЗАБАРВЛЕННЯ ТЕКСТУ. Литвинов М.Г., студент магістратури ПІ. Науковий керівник: д.т.н., проф. Єрохін А.Л.....42
7. DEVELOPMENT AND RESEARCH METHODS OF FORMATION AND FUNCTIONING OF INTELLECTUAL TEAM. Ponomarova S.V., graduate student ST Department. Academic adviser: Ph.D., Assoc. Prof. Kalyta N.I.....50
8. INVESTIGATION OF RECOGNITION METHODS OF A PERSON'S FATIGUE USING A 3D CAMERA. Bakhmet A.G., graduate student SE Department. Academic adviser: Ph.D., Assoc. Prof. Nazarov O.S.....63
9. ПРОГРАМНА СИСТЕМА КЕРУВАННЯ РОЗУМНИМ ЛІЖКОМ «BEDDEST». Єлісєєва М.С., студент магістратури ПІ. Науковий керівник: к.т.н., доц. Мазурова О.О.....73
10. САМООРГАНІЗОВНА ІНКРЕМЕНТНА НЕЙРОННА МЕРЕЖА ДЛЯ КЛАСТЕРУВАННЯ МАСИВІВ ДАНИХ. Іванова Є.В., студентка групи КН-14-5. Науковий керівник: к.т.н., ст. викл. каф.ШІ Дейнеко А.О. 86

ДОСЛІДЖЕННЯ МЕТОДІВ ТА ПРОГРАМНИХ ЗАСОБІВ АНАЛІЗУ НОВИННИХ СТРІЧОК В СОЦІАЛЬНИХ МЕРЕЖАХ

Сердюк О.О., студент магістратури ПІ,
e-mail: oksana_serdiuk@nure.ua.
Науковий керівник: д.т.н., проф. Єрохін А.Л.
Харківський національний університет радіоелектроніки

The article is devoted to the methods of text mining, analyzing text and weighting words in a document in order to retrieve words those are more semantically focused. That helps to retrieve information about it main topic, tone and content.

Кожна людина зараз проводить переважну частину часу в соціальних мережах. За статистикою люди проводять у соціальних мережах мінімум годину часу. За цей час в мережу потрапляє надзвичайна кількість необробленої інформації, яка здатна впливати на свідомість людей, на їхні думки, точки зору. Аналіз усього потоку текстової інформації дозволив би вирішити декілька проблем в різних сферах.

На основі результатів аналізу тексту можна зробити висновок про його тематику, основну думку, основний мотив, тональність написаного. На рівні держави і бізнесу важливим результатом є можливість визначити найбільш обговорювані теми, загальний настрій людей у певний період з перелічених показників, або навіть прогнозування можливих реакцій на певні події. Також, у сфері соціальних мереж для кінцевих користувачів було б корисно мати фільтр з великою кількістю налаштувань, що виконує пошук згідно з даними показниками.

Для вирішення перелічених проблем можна скористатися одним методом: інтелектуальним аналізом тексту. Схема інтелектуального аналізу тексту представлена на рисунку 1. Інтелектуальний аналіз тексту призначений для обробки неструктурованого набору інформації з метою вилучення з нього корисних даних. Наприклад, він може бути використаний для виділення основної теми тексту, порівняння двох текстів на

схожість. У нашому випадку він стане в нагоді для визначення основної думки тексту і його тональності [1].

Інтелектуальний аналіз тексту обробляє текст перетворюючи його в значимі індекси, які можуть використовуватися в подальшому, наприклад, в машинному навчанні [2]. Кожне слово в тексті повинно бути проіндексовано і пораховано для складання таблиці матриці частот, яка відображає частоту вживання того чи іншого слова.

Далі побудована таблиця може бути оброблена для видалення незначущих слів, артиклів, з'єднання однакових слів, які були вжиті в різних формах (число, час, відмінок). Один з варіантів нормалізації називається стемінг. Необхідно звернути увагу, що процес стемінгу специфічний для кожної мови, так як кожна мова має свої слова, особливості їх вживання та граматичні конструкції.

Стемінг позначає процес знаходження основи слова для заданого вихідного слова так, щоб різні граматичні форми одного слова були зараховані як одне [4]. Наприклад, «читає» і «читав» має бути проіндексовано як одне слово. Тут важливо знайти синоніми і словосполучення, які мають значення відмінне від того, якби ці ж слова, присутні у фразі, були вжиті окремо. Також має сенс виключити з статистики рідкісні слова, які швидше за все не відбивають суть тексту.

Рисунок 1 – Схема інтелектуального аналізу тексту.

Зазвичай стеммером користуються для пошуку тексту з імітацією врахування морфології. Під імітацією розуміють непереборно велику кількість помилок і нерелевантних результатів, які виникають, якщо застосовувати тільки стеммер [5]. У

слов'янських мовах джерелом помилок при стемінгу є всілякі зміни кореня слова. Наочно проблеми, пов'язані з використанням стеммер, можна продемонструвати для іменника «кішка». Родовий відмінок множини має форму «кішок». Таким чином, найдовший спільний префікс всіх форм іменника «кішка» - це «кіш». Якщо виконати пошук тексту по цьому префіксу, то в результатах з може опинитися слово «Кішма», що є назвою річки. Або можливий ще один варіант помилки для цього слова, коли стеммер обирає префікс «кішк», що унеможливлює появу слова «кішок» у результатах.

Таким чином, можна виділити наступні помилки при стемінгу. Стем дає занадто велике узагальнення і тому зіставляється з граматичними формами більш ніж однієї словникової статті. Це найчисленніша група помилок стемінгу. Наприклад, якщо в стемінге братиме участь слово «вам», то в подальшому знайти певний текст дасть збіг зі словом «вампір». Компенсація помилок такого роду успішно виконується або введенням списку стоп-слів, або більш якісно – лематизатором або флексером.

Наступні помилки, пов'язані з тим, що усічення форми дає занадто довгий стем, що не зіставляється з деякими граматичними формами цього ж слова [6]. До таких помилок призводить прагнення розробника стеммеру знайти компроміс з помилками першого роду в разі, коли при словотворенні змінюється основа слова. Такі слова є навіть у англійській мові, що багата частими словозмінами (неправильні дієслова). У українській мові випадки зміни основи слова навіть не є підставою для віднесення слова до групи неправильних, настільки часте це явище.

Помилкою стемінгу третього роду є відсутність можливості побудувати стем через зміни в корені слова, яке залишає єдину букву в стемі. Або модель словозміни має на увазі використання префіксів. Приклад для першого випадку слова «шити», «вити», «лити», «пити», які мають форми «шию», «в'ють», «плє», «п'ю». Дані слова мають корінь, що складається з однієї літери. Другий випадок виникає в рамках граматичного словника для вищого ступеня прикметників і прислівників в українській мові – наприклад «побільше» як більш висока форма порівняння для прислівника «більше».

Для ефективнішого текстового аналізу доповненням до стеммінгу може виступати лематизація [7]. Лематизацією

називається перетворення слова в словниковий вид або лемму. Даний метод використовується в алгоритмах пошукових систем при індексації сторінок. Процес дає можливість зберігання даних сторінки набором слів в індексі для зручної схематизації файлів. Це дозволяє прискорити індексацію і сформувати більш чітку відповідь на пошуковий запит, так як скорочену форму слова пошукова система аналізує швидше.

Лематизації передбачає порівняння основ слів, однак вона проводиться з урахуванням частин мови, до якої відносяться словоформи. Наприклад, стеммер для «читати», «читав», «читач», «читачі» з коренем «чит» і зробить висновок, що це одне і те саме слово. У той час як лематизатор виділить окремо слова «читати» і «читав» як форми слова «читати», і слова «читач» і «читачі» як форми слова «читач». Під лемою мається на увазі лексема, завдання лематизації ототожнити словоформи, співвідносні з однією лексемою.

Після побудови таблиці унікальних слів її можна використовувати для кластеризації, класифікації, виділення важливих слів, що несуть певне смислове навантаження, або термінів [8].

Постає питання як зрівняти смислову вагу слів посеред інших, адже просто визначити частоту вживання недостатньо для більш ефективного рішення. Тому для визначення смислової ваги слова використовують наступні терміни.

TF (term frequency) – частота вживання слова в документі. Оскільки кожен документ відрізняється за довжиною, можливо, що слово у довгих документах з'являється набагато більше разів, ніж у коротших. Таким чином, term frequency часто ділиться на довжину документу (тобто загальне число слів у документі) як спосіб нормалізації:

$$tf_{i,j} = \frac{n_i}{\sum_j n_j},$$

де n_i – кількість разів, коли слово i з'являється в документі j, n_j – загальна кількість слів у документі j.

DF (document frequency) – число документів у яких вживається і-е слово. Під час обчислення TF всі слова вважаються однаково важливими. Проте відомо, що певні слова, такі як "є", "з", "що", можуть з'являтися багато разів, але мають мало значення. Таким чином, нам потрібно зменшити важливість таких слів, збільшуючи

вагу слів, що зустрічаються не так часто, шляхом обчислення наступного:

$$df_{i,j} = \frac{d_i}{\sum_j d_j},$$

де n_i – кількість документів, у яких з'являється слово i, d_j – загальна кількість документів.

Інформація, що визначається TF (частотою слова), дає зрозуміти на скільки важливим є слово для даного тексту. Чим більше частота слова, тим більш ймовірно, що це слово добре описує зміст тесту.

Але далеко не завжди, якщо в одному тексті слово зустрічається один раз, а в іншому 3 рази, значить, що в другому воно краще описує вміст. Тому виконуються додаткові обчислення для того, щоб послабити вплив цієї частоти на результат. Можна виконати одне з наступних обчислень:

$f(tf) = \sqrt{tf}$

або

$f(tf) = 1 + log(tf)$, **для** $tf > 0$, **де** tf – **частота вживання слова**.

Наприклад, якщо слово «машина» зустрічається у тексті 3 рази, то $tf = \sqrt{3}$ або tf =1 + log 3 краще відображають важливість слова з частотою слова 3, ніж просто кількість разів. Документ з частотою слова $tf = \sqrt{3}$ або tf =1 + log 3 буде дещо важливіший ніж документ з однією появою слова ($tf = \sqrt{1}$ або tf =1 + log 1), але не втричі важливіший.

DF (частота документу) може бути інтерпретована як індикатор інформативності. За допомогою частоти документу, можна нівелювати ще одну проблему. Припустимо в тексті слово «заняття» зустрічається 10 разів, а слово «парламент» зустрічається тільки 3 рази. Хоча, як раз слово «парламент» краще описує і відображає зміст тексту. Справа в тому, що слово «заняття» може вживатися кожен раз в новому контексті, яке ніяк не стосується теми тексту. Наприклад, «заняття у школі», «заняття як хобі». А слово «парламент» вузькоспеціалізоване і вживається у текстах переважно політичної тематики.

Це пояснюється тим, що семантично сфокусоване слово зазвичай вживається у тексті кілька разів, якщо вживається взагалі. А семантично несфокусоване слово розподілене по всьому тексту. Також, у разі якщо у тексті було вжито семантично сфокусоване

слово, велика ймовірність, що це слово буде вжите ще кілька разів [9].

Поєднати частоту слова та частоту документу в одну вагову одиницю і-го слова і j-го документа можна за допомогою наступної формули, що називається Inverse Document Frequency (IDF):

$$w(i,j) = \begin{cases} 0\text{, якщо } tf_{ij} = 0 \\ \left(1 + log(tf_{ij})\right) log \frac{N}{df_i}\text{, якщо } tf_{ij} \geq 1' \end{cases}$$

де $w(i,j)$ – вага і-го слова в j-ому документі, N – загальна кількість документів, df_i – кількість документів, які використовують і-е слово, tf_{ij} – частота вживання і-го слова в j-ому документі.

Отже, можна побачити, що ця формула включає в себе як звуження простих частот слів за допомогою функції логарифму, а також включає вагових фактор, який прирівнює до 0, якщо слово зустрічається у всіх документах ($log(N/N = 1) = 0$), і до максимального значення, коли слово зустрічається лише в одному документі ($log(N/1) = log(N)$). Цілком легко зрозуміти, як ця трансформація створює індекси, які відображають відносні частоти слів, а також їх семантичні особливості в документах, що були включені до аналізу.

Література.

1. T. Mikolov, K. Chen, G. Corrado, J. Dean, «Efficient Estimation of Word Representations in Vector Space» In Proc. of Workshop at ICLR, 2013.

2. Hill T., Lewicki P. Statistics: Methods and Applications. Comprehensive Reference for Science, Industry, and Data Mining / T. Hill, P. Lewicki –Tulsa: StatSoft, Inc, 2006. – 832 с.

3. Text Mining (Big Data, Unstructured Data). [Електронний ресурс] / TIBCO Statistica – Режим доступу: http://www.statsoft.com/Textbook/Text-Mining – 10.03.2018 р. – Загол. з екрану.

4. Большакова Е.И., Воронцов К.В., Ефремова Н.Э., Клышинский Э.С., Лукашевич Н.В., Сапин А.С. Автоматическая обработка текстов на естественном языке и анализ данных : учеб. пособие / Е.И. Большакова, К.В. Воронцов, Н.Э. Ефремова, Э.С.

Serdiuk Oksana Olehivna
Student, Department of Computer Science,
Kharkiv National University of Radio Electronics
Kharkiv, Ukraine; E-mail: oksana.serdiuk@nure.ua

TEXT CLASSIFICATION STAGES FOR SOCIAL MEDIA

The classification is used for dividing documents into a fixed number of categories. Each document may belong to one or more categories or to none. Using machine learning, it is possible to delegate this operation to a program. In order to do this, a programmer should train the program on a training dataset that is already tagged with categories. And only after it, it will be able to perform classification by itself.

There are some issues that can appear and reduce the effectiveness of text mining. Therefore, the first major step in the analysis of the text is preparation [1].

The first stage of pre-processing data is tokenization, i.e. dividing words into tokens [2]. Token represents the smallest unit of text – a word. However, this approach has many disadvantages. One of the options is splitting by a non- non-alphanumeric character. In this case, there is a huge disadvantage because it can split the word "O'Reilly" into "O" and "Reilly", that is completely wrong so as it is one word. Another option is splitting by a whitespace, but it may have some problems in languages such as Chinese, Japanese, Thai.

Tokenization is considered as the easiest stage of text analysis and one of the most uninteresting tasks. However, mistakes made in this phase will affect the following phases and cause problems.

In the grammar of any language, there are rules for words inflection to express different grammatical categories such as tension, time, personality,

Клышинский, Н.В. Лукашевич, А.С. Сапин — М.: Изд-во НИУ ВШЭ, 2017. — 269 с.

5. Леонтьева Н. Н. Автоматическое понимание текстов: Системы, модели, ресурсы: Учебное пособие / Н. Н. Леонтьева — М.: Академия, 2006.

6. Плунгян В.А. Общая морфология. Введение в проблематику. / В.А. Плунгян — М.: Едиториал УРСС. — 2003. 384 с.

7. Лемматизация. [Електронний ресурс] / Компьютерная грамматика русского языка: лексика, морфология, синтаксис – Режим доступу: http://www.solarix.ru/for_developers/api/lemmatization.shtml – 10.03.2018 р. – Загол. з екрану.

8. Клышинский Э.С. Начальные этапы анализа текста // Автоматическая обработка текстов на естественном языке и компьютерная лингвистика: учеб. пособие / Э.С. Клышинский — М.: МИЭМ, 2011.

9. Manning C. D., Schiitze H. Foundations of Statistical Natural Language Processing/ C. D. Manning, H. Schiitze– London: The MIT Press, 1999. – 704 с.

quantity, gender and mood. Several options that are available are stemming and lemmattisation. The main idea of both is to reduce words to the stem or to the base word. Stemming is the easiest approach from these two. It is a process of reducing a word to its stem, even if there is no such root in the dictionary. A stem is a part of a word that is a subject to inflection. Stemming algorithms are usually based on rules. You can view them as a heuristic process that sorts the ends of words. The word is considered and executed through a number of conditions that determine how to cut it. However, since the stemming is usually based on heuristics, it is far from perfect. In fact, it usually suffers from two problems, in particular: excessive reduction and lack of stemming. Excessive reduction occurs when a word is being reduced more than it is necessary. It can lead to stems that are not meaningful and the meaning of the whole text will be lost or confused. Or it may lead to the situation when completely different words are reduced to the same stems. Lack of stemming occurs when there are several words that are in fact forms of each other, but in the result of stemming, they are reduced to different stems.

The most famous algorithms for stemming are the Porter's stemming and the NLTK Snowball. The Porter Stemmer was developed in 1980s, and its main task is to eliminate the common endings of words so that they can be transformed into a common form [3]. This is a quite simple task, so this algorithm does not develop further. The Snowball algorithm, also known as the Porter2 algorithm, is considered to be better than the Porter Stemmer. It is based on the problems that were discovered when working with the first algorithm. The Lancaster Stemmer is the most aggressive one. When using it in NLTK, it is possible to add your own rules to this algorithm. The disadvantage of this algorithm is its excessive aggressiveness. It can lead to too short stems.

Lemmatization reduces the words properly, ensuring that the root word is correct and exists in the dictionary. In lemmatization, the root word is called a lemma. This is a canonical form or a vocabulary of the word. Since lematization returns the actual word of the language, it is used where it is necessary to get the correct words. Lemmatization has more nuances than stemming, and therefore requires more efforts [4].

Consequently, as a result of a stemming or lematization, documents are transformed into a list of indexed words, which will then be processed. This array of words may be converted into a dictionary that is a collection of key-value pairs, where the key is a lemma, and the value is the frequency of word.

It makes sense to take into account only the words that occur most often. Term Frequency (TF) is the ratio of the number of word appearances in a document to the total number of words in the document [5]. The calculation of TF is presented in formula 2.1.

$$TF(w) = \frac{n(w)}{n(d)}, \quad (1)$$

where w – a specific word, $n(w)$ – an amount of the word appearances, $n(d)$ – an amount of all words in the document.

IDF (inverse document frequency) is used for analysing a set of documents. It helps to reduce the weight of widely used words. The more documents contain a specific word, the lower the IDF will be. The IDF calculation is presented in formula 2.

$$IDF(w) = log\frac{n}{DF(w)}, \quad (2)$$

where w – a specific word, n – an amount of documents to analyze, $DF(w)$ – an amount of documents which contains the word w.

TF-IDF is a union of two measurements. It increases the weight of words that are used most rarely within all documents and most often within one document. The calculation of TF-IDF is presented in formula 3.

$$TFIDF(w) = TF(w) \cdot IDF(w) \quad (3)$$

TF-IDF can be used with SVM algorithm [6]. Support Vector Machines is a classification algorithm that works on already vectorized data and finds the maximum difference that separates the hyperplane. SVM is used for tasks such as category assignment, spam detection, and sentiment analysis.

TF-IDF can also be combined with Word2Vec algorithm [7]. It transforms a word into unique vector that can be manipulated in different ways. For example, "king" + (" woman "-" man "). The resulting vector turns out to be close to the vector corresponding to the term "queen".

The summary of the mentioned above is the following. This algorithm has the advantages of accuracy and efficiency on small data sets. It can also be more effective because it uses a training set of data. Negative aspects of SVM are unsuitable for large data sets, as time increases for training and processing, there is less efficiency in data sets with unclear division into classes. Word2Vec, in turn, calculates words importance and does not need a lot of training, except building a model, and always keeps in mind the words context which is more suitable for the news feed analysis.

References

1. A General Approach to Preprocessing Text Data // Machine Learning, Data Science, Big Data, Analytics, AI. URL: https://www.kdnuggets.com/2017/12/general-approach-preprocessing-text-data.html (Accessed: 29.03.2019).

2. Bullinaria, John A., and Joseph P. Levy. Extracting semantic representations from word cooccurrence statistics: stop-lists, stemming, and SVD. – Behavior research methods 44.3, 2012. – C. 890-907.

3. The Porter stemming algorithm // Snowball. URL: http://snowball.tartarus.org/algorithms/porter/stemmer.html/ (Accessed: 29.03.2019).

4. Жердева М.В., Артюшенко В.М. Стемминг и лемматизация в lucene. Net // Вестник МГУЛ – Лесной вестник. 2016. №3. URL: https://cyberleninka.ru/article/n/stemming-i-lemmatizatsiya-v-lucene-net (Accessed: 29.03.2019).

5. The TF-IDF Algorithm Explained // The One and Only Technical SEO House | Onely. URL: https://www.onely.com/blog/what-is-tf-idf/ (Accessed: 29.03.2019).

6. Support vector machines and machine learning on documents // Meet Guru99 – The Stanford Natural Language Processing Group. URL: https://nlp.stanford.edu/IR-book/html/htmledition/support-vector-machines-and-machine-learning-on-documents-1.html (Accessed: 29.03.2019).

7. Word Embedding Tutorial: word2vec using Gensim // Meet Guru99 – Free Training Tutorials & Video for IT Courses. URL: https://www.guru99.com/word-embedding-word2vec.html (Accessed: 29.03.2019).

ДОДАТОК Г

Диск (CD-R)