Universidad de Chile
Departamento de Ciencias de la Computación

Procesamiento de Lenguaje Natural

Felipe Bravo-Márquez

Índice general

1. Introducción **1**
- 1.1. PLN y Lingüística Computacional . . . 1
- 1.2. Niveles de descripción lingüística . . . 3
  - 1.2.1. Fonética . . . 3
  - 1.2.2. Fonología . . . 3
  - 1.2.3. Morfología . . . 3
  - 1.2.4. Sintaxis . . . 4
  - 1.2.5. Semántica . . . 4
  - 1.2.6. Pragmática . . . 5
- 1.3. Procesamiento del Lenguaje Natural y Aprendizaje Automático 6
- 1.4. Desafíos del Lenguaje . . . 7
- 1.5. Ejemplo de tareas NLP . . . 7
  - 1.5.1. Lingüística y Procesamiento del Lenguaje Natural (PNL) 9
- 1.6. Desafíos en el Procesamiento del Lenguaje Natural (PNL) . . . 10
- 1.7. Estudio de caso: Clasificación de sentimientos en tweets . . . 11
- 1.8. Ingeniería de características y Aprendizaje Profundo . . . 12
- 1.9. Historia . . . 13
- 1.10. Conclusiones . . . 14

2. Modelo de Espacio Vectorial y Recuperación de Información **15**
- 2.1. Tokens y Tipos . . . 15
  - 2.1.1. Ley de Zipf . . . 17
  - 2.1.2. Listas de publicaciones y el índice invertido . . . 18
  - 2.1.3. Motores de búsqueda web . . . 18
- 2.2. El modelo de espacio vectorial . . . 19
  - 2.2.1. Similitud entre vectores . . . 21
- 2.3. Agrupamiento de Documentos . . . 22
  - 2.3.1. K-Means . . . 23
- 2.4. Conclusiones y Conceptos Adicionales . . . 23

3. Modelos de Lenguaje Probabilísticos **25**
- 3.1. The Language Modeling Problem . . . 25
  - 3.1.1. Why would we want to do this? . . . 26
  - 3.1.2. Language Models are Generative . . . 27

3.1.3. A Naive Method . . . 27
3.2. Markov Processes . . . 28
3.2.1. Modeling Variable Length Sequences . . . 29
3.3. Trigram Language Models . . . 29
3.3.1. The Trigram Estimation Problem . . . 29
3.4. Evaluating a Language Model: Perplexity . . . 30
3.5. Some History . . . 31
3.6. The Bias-Variance Trade-Off . . . 31
3.7. Linear Interpolation . . . 32
3.8. Estimating λ Values . . . 32
3.9. Discounting Methods . . . 33
3.9.1. Katz Back-Off Models (Bigrams) . . . 34
3.10. Summary . . . 34

4. Text Classification and Naïve Bayes **35**
4.1. Text Classification: Definition . . . 37
4.1.1. Classification Methods: Hand-coded rules . . . 37
4.1.2. Classification Methods: Supervised Machine Learning . 37
4.1.3. Supervised Learning Problems . . . 38
4.1.4. Generative Models . . . 38
4.1.5. Classification with Generative Models . . . 38
4.2. Naive Bayes Intuition . . . 38
4.2.1. Bayes' Rule Applied to Documents and Classes . . . 38
4.3. Naive Bayes Classifier . . . 39
4.3.1. Multinomial Naive Bayes Independence Assumptions . 40
4.3.2. Multinomial Naive Bayes Classifier . . . 40
4.3.3. Applying Multinomial Naive Bayes Classifiers to Text Classification . . . 40
4.3.4. Problems with Multiplying Lots of Probabilities . . . 40
4.3.5. Learning the Multinomial Naive Bayes Model . . . 41
4.3.6. Parameter Estimation . . . 41
4.3.7. Zero Probabilities and the Issue of Unseen Words . . . 41
4.3.8. Laplace (Add-1) Smoothing for Naïve Bayes . . . 42
4.3.9. Multinomial Naïve Bayes: Learning . . . 42
4.3.10. Unknown Words . . . 42
4.3.11. Stop Words . . . 43
4.4. Worked Sentiment Example . . . 43
4.5. Naive Bayes as a Language Model . . . 43
4.6. Evaluation . . . 44
4.6.1. The 2-by-2 Confusion Matrix . . . 44
4.6.2. Evaluation: Accuracy . . . 45
4.6.3. Evaluation: Precision and Recall . . . 45
4.6.4. Why Precision and Recall? . . . 45
4.6.5. A Combined Measure: F-measure . . . 45
4.6.6. Development Test Sets ("Devsets") . . . 46
4.6.7. Cross-validation: Multiple Splits . . . 46

4.6.8. Confusion Matrix for 3-class classification 47
4.6.9. Macroaveraging and Microaveraging 47

Índice de cuadros

2.1. Matriz tf-idf 22

Índice de figuras

1.1. Reconocimiento de Entidades Nombradas 1

2.1. Ley de Zipf 18
2.2. Índice invertido 18
2.3. Los diversos componentes de un motor de búsqueda web [Manning et al., 2008]. 19
2.4. Similitud del coseno. 21
2.5. Conjunto de documentos donde los grupos se pueden identificar claramente. 23
2.6. Algoritmo K-means 24

4.1. James Madison 36
4.2. Alexander Hamilton 36

Capítulo 1

Introducción

El volumen de datos textuales digitalizados que se genera cada día es enorme (por ejemplo, la web, redes sociales, registros médicos, libros digitalizados). Por lo tanto, también crece la necesidad de traducir, analizar y gestionar esta avalancha de palabras y texto.

El procesamiento del lenguaje natural (PLN) es el campo que se encarga de diseñar métodos y algoritmos que toman como entrada o producen como salida datos de **lenguaje natural** no estructurado [Goldberg, 2017]. El PLN se centra en el diseño y análisis de algoritmos computacionales y representaciones para procesar el lenguaje humano [Eisenstein, 2018].

Una tarea común de PLN es el Reconocimiento de Entidades Nombradas (NER, por sus siglas en inglés). Por ejemplo:

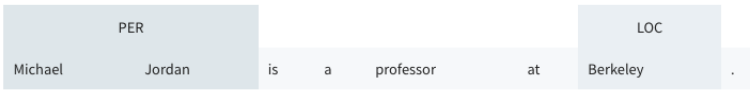


Figura 1.1: Reconocimiento de Entidades Nombradas

El lenguaje humano es altamente ambiguo, como en las frases: Çomí pizza con amigos", Çomí pizza con aceitunas.º Çomí pizza con un tenedor". Además, el lenguaje está en constante cambio y evolución, como ocurre con los hashtags en Twitter.

1.1. PLN y Lingüística Computacional

PLN suele confundirse con otra disciplina hermana llamada Lingüística Computacional (LC). Si bien ambas están estrechamente relacionadas, tienen un foco distinto. La LC busca responder preguntas fundamentales sobre el lenguaje mediante el uso de la computación, es decir, cómo entendemos el lengua-

je, cómo producimos lenguaje o cómo aprendemos lenguaje. Mientras que en PLN el foco está en resolver problemas específicos, tales como las transcripción automática del habla, la traducción automática, la extracción de información de documentos y el análisis de opiniones en redes sociales. Es importante señalar que en PLN, el éxito de una solución se mide en base métricas concretas (Ej: qué tan similar es la traducción automática a una hecha por un humano) independientemente si el modelo hace uso de alguna teoría lingüística.

El procesamiento del lenguaje natural (PLN) desarrolla métodos para resolver problemas prácticos relacionados con el lenguaje [Johnson, 2014].

Algunos ejemplos son:

- Reconocimiento automático del habla.
- Traducción automática.
- Extracción de información de documentos.

La lingüística computacional (LC) estudia los procesos computacionales subyacentes al lenguaje (humano).

- ¿Cómo comprendemos el lenguaje?
- ¿Cómo producimos el lenguaje?
- ¿Cómo aprendemos el lenguaje?

El PLN y la LC utilizan métodos y modelos similares.

Aunque existe una superposición sustancial, hay una diferencia importante en el enfoque. La LC se centra en la lingüística respaldada por métodos computacionales (similar a la biología computacional o la astronomía computacional). En lingüística, el lenguaje es el objeto de estudio. El PLN se centra en resolver tareas bien definidas relacionadas con el lenguaje humano (como la traducción, la respuesta a consultas, las conversaciones). Si bien los conocimientos lingüísticos fundamentales pueden ser cruciales para realizar estas tareas, el éxito se mide en función de si y cómo se logra el objetivo (según una métrica de evaluación) [Eisenstein, 2018].

El procesamiento del lenguaje natural y la lingüística computacional están estrechamente relacionados y se superponen en muchos aspectos. Ambos campos utilizan métodos y modelos similares para abordar problemas relacionados con el lenguaje humano. Sin embargo, la diferencia principal radica en el enfoque: la lingüística computacional se centra en la lingüística respaldada por métodos computacionales, mientras que el procesamiento del lenguaje natural se centra en resolver tareas prácticas relacionadas con el lenguaje. Ambos campos son fundamentales para comprender y aprovechar el poder del lenguaje humano en la era digital.

1.2. Niveles de descripción lingüística

El campo de la **descripción lingüística** abarca diferentes niveles:

- **Fonética y fonología:** estudio de los sonidos del habla.
- **Morfología:** estudio de la estructura de las palabras.
- **Sintaxis:** estudio de la estructura de las oraciones.
- **Semántica:** estudio del significado de las palabras y oraciones.
- **Pragmática:** estudio del uso del lenguaje en el contexto.

El PLN puede abordar tareas en cada uno de estos niveles, pero a menudo se enfoca en niveles más altos de representación y comprensión.

1.2.1. Fonética

La fonética es la rama de la lingüística que se ocupa del estudio de los sonidos del lenguaje. Examina los órganos utilizados en la producción de sonidos, como la boca, la lengua, la garganta, la nariz, los labios y el paladar. Los sonidos del lenguaje se dividen en vocales y consonantes. Las vocales se producen con poca restricción del flujo de aire desde los pulmones, mientras que las consonantes implican alguna restricción o cierre en el tracto vocal [Johnson, 2014, Fromkin et al., 2018]. Además, el Alfabeto Fonético Internacional (AFI) proporciona una notación alfabética para representar los sonidos fonéticos.

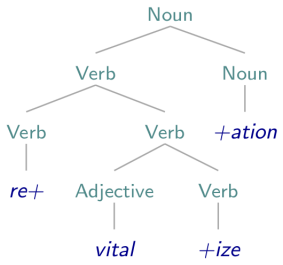
1.2.2. Fonología

La fonología se centra en el estudio de cómo los sonidos del habla forman patrones y construyen significado. Los fonemas son las unidades básicas de sonido que diferencian el significado de las palabras. Por ejemplo, en inglés, la "pɔ la "b"son fonemas distintos porque cambian el significado de las palabras en las que se encuentran. La fonología también examina las variaciones en la pronunciación de los sonidos en diferentes contextos y dialectos [Fromkin et al., 2018].

1.2.3. Morfología

La morfología se ocupa del estudio de la estructura interna de las palabras. Los morfemas son las unidades mínimas de significado que componen las palabras. Por ejemplo, en la palabra "deshacer", los morfemas son "des-", "hacerɔ er". La morfología también se interesa por los procesos de formación de palabras, como la derivación, donde se agregan prefijos o sufijos a una palabra existente para formar una nueva palabra con un significado diferente [Johnson, 2014].

- La morfología estudia la estructura de las palabras (por ejemplo, re+estructur+ando, in+olvid+able) [Johnson, 2014]
- Morfema: el término lingüístico para la unidad más elemental de forma gramatical [Fromkin et al., 2018]. Por ejemplo, morfología = morf + ología (la ciencia de).
- Morfología derivativa: proceso de formar una nueva palabra a partir de una palabra existente, a menudo mediante la adición de un prefijo o sufijo.
- La morfología derivativa exhibe una estructura jerárquica. Ejemplo: re+vital+iz+ación



- El sufijo generalmente determina la categoría sintáctica (part-of-speech) de la palabra derivada.

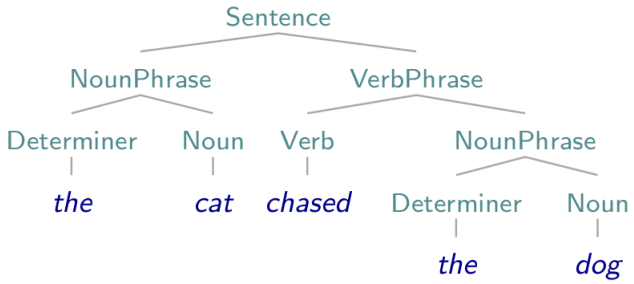
1.2.4. Sintaxis

La sintaxis es el estudio de cómo las palabras se combinan para formar frases y oraciones gramaticales. Examina las reglas y estructuras que determinan la organización de las palabras en una oración y cómo influyen en el significado. La sintaxis también se ocupa de la relación entre las palabras y las funciones que desempeñan dentro de una oración. Por ejemplo, en la oración .El perro persigue al gato", .el perro.es el sujeto, "persigue.es el verbo y .al gato.es el complemento directo [Johnson, 2014].

- La sintaxis estudia las formas en que las palabras se combinan para formar frases y oraciones [Johnson, 2014]
- El análisis sintáctico ayuda a identificar **quién hizo qué a quién**, un paso clave para comprender una oración.

1.2.5. Semántica

La semántica es el estudio del significado de las palabras, frases y oraciones, examinando cómo se construye e interpreta este significado en el contexto del lenguaje. Además, la semántica se interesa por los roles semánticos, que



indican la función de cada entidad en una oración. Por ejemplo, en la oración .El niño cortó la cuerda con una navaja", .el niño.es el agente, "la cuerda.es el tema y üna navaja.es el instrumento [Johnson, 2014].

La semántica se enfoca en el significado de las palabras, frases y oraciones. Estudia cómo se construye e interpreta este significado en el contexto del lenguaje. Además, dentro de la semántica, se analizan los roles semánticos, los cuales indican la función que desempeña cada entidad en una oración. Por ejemplo, en la oración .El niño cortó la cuerda con una navaja", se identifican distintos roles semánticos: .el niñoçomo el agente, "la cuerdaçomo el tema y üna navajaçomo el instrumento utilizado [Johnson, 2014].

En resumen:

- La semántica estudia el significado de las palabras, frases y oraciones [Johnson, 2014].
- Dentro de la semántica, se analizan los roles semánticos, que indican el papel desempeñado por cada entidad en una oración.
- Algunos ejemplos de roles semánticos son: **agente** (la entidad que realiza la acción), **tema** (la entidad involucrada en la acción) y **instrumento** (otra entidad utilizada por el agente para llevar a cabo la acción).
- En la oración .El niño cortó la cuerda con una navaja", se puede identificar el agente como **el niño**, el tema como **la cuerda** y el instrumento como **una navaja**.
- Además de los roles semánticos, la semántica también abarca las relaciones léxicas, que son las relaciones entre diferentes palabras [Yule, 2016].
- Algunos ejemplos de relaciones léxicas incluyen la sinonimia (conceal/hide), la antonimia (shallow/deep) y la hiponimia (perro/animal).

1.2.6. Pragmática

La pragmática se centra en cómo el contexto influye en la interpretación y el significado de las expresiones lingüísticas. Examina cómo se utilizan las expre-

siones lingüísticas en situaciones reales y cómo los hablantes interpretan el significado implícito. Por ejemplo, la oración "Hace frío aquí"puede interpretarse como una sugerencia implícita de cerrar las ventanas [Fromkin et al., 2018].

1.3. Procesamiento del Lenguaje Natural y Aprendizaje Automático

Comprender y producir el lenguaje computacionalmente es extremadamente complejo. La tecnología más exitosa actualmente para abordar PLN es el aprendizaje automático supervisado que consiste en una familia de algoritmos que "aprenden" a construir la respuesta del problema en cuestión en base a encontrar patrones en datos de entrenamiento etiquetados. Por ejemplo, si queremos tener un modelo que nos diga si un tweet tiene un sentimiento positivo o negativo respecto a un producto, primero necesito etiquetar manualmente un conjunto de tweets con su sentimiento asociado. Luego debo entrenar un algoritmo de aprendizaje sobre estos datos para poder predecir de manera automática el sentimiento asociado a tweets desconocidos. Como se podrán imaginar, el etiquetado de datos es una parte fundamental de la solución y puede ser un proceso muy costoso, especialmente cuando se requiere conocimiento especializado para definir la etiqueta.

Aunque los seres humanos somos grandes usuarios del lenguaje, también somos muy malos para comprender y describir formalmente las reglas que rigen el lenguaje.

Entender y producir lenguaje utilizando computadoras es altamente desafiante. Los métodos más conocidos para lidiar con datos de lenguaje se basan en el aprendizaje automático supervisado.

El aprendizaje automático supervisado consiste en intentar inferir patrones y regularidades a partir de un conjunto de pares de entrada y salida preanotados (también conocido como conjunto de datos de entrenamiento).

Conjunto de Datos de Entrenamiento: Datos de NER CoNLL-2003 Cada línea contiene un token, una etiqueta de parte de la oración, una etiqueta de sintagma y una etiqueta de entidad nombrada.

```
U.N.          NNP  I-NP  I-ORG
official      NN   I-NP  O
Ekeus         NNP  I-NP  I-PER
heads         VBZ  I-VP  O
for           IN   I-PP  O
Baghdad       NNP  I-NP  I-LOC
.             .    O     O
```

[1]Fuente: `https://www.clips.uantwerpen.be/conll2003/ner/`

1.4. Desafíos del Lenguaje

Existen tres propiedades desafiantes del lenguaje: la discreción, la composicionalidad y la dispersión.

Discreción: no podemos inferir la relación entre dos palabras a partir de las letras que las componen (por ejemplo, hamburguesa y pizza).

Composicionalidad: el significado de una oración va más allá del significado individual de sus palabras.

Dispersión: la forma en que las palabras (símbolos discretos) pueden combinarse para formar significados es prácticamente infinita.

1.5. Ejemplo de tareas NLP

Clasificación de temas La clasificación de temas es una tarea de Procesamiento del Lenguaje Natural (PLN) en la cual se asigna a un documento una de varias categorías, como deportes, política, cotilleos o economía. Las palabras presentes en los documentos brindan pistas importantes sobre su tema. Sin embargo, redactar reglas para esta tarea es un desafío debido a la complejidad del lenguaje. La anotación de datos, en la cual los lectores clasifican los documentos por temas, puede ayudar a generar conjuntos de datos de entrenamiento para algoritmos de aprendizaje automático supervisado. Estos algoritmos aprenden patrones de uso de palabras que facilitan la categorización de los documentos.

- Clasificar un documento en una de las cuatro categorías: Deportes, Política, Cotilleos y Economía.
- Las palabras en los documentos proporcionan indicios muy sólidos.
- ¿Qué palabras brindan qué indicios?
- Elaborar reglas para esta tarea resulta bastante desafiante.
- No obstante, los lectores pueden categorizar fácilmente varios documentos según su tema (anotación de datos).
- Un algoritmo de aprendizaje automático supervisado puede identificar los patrones de uso de palabras que ayudan a categorizar los documentos.

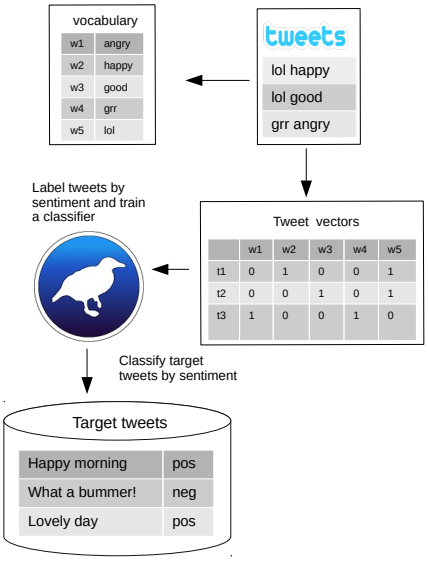
Análisis de Sentimiento El análisis de sentimientos se refiere a la aplicación de técnicas de Procesamiento del Lenguaje Natural (PLN) para identificar y extraer información subjetiva de conjuntos de datos textuales. Un desafío común en el análisis de sentimientos es la clasificación de la polaridad a nivel de mensaje (MPC), donde las frases se clasifican automáticamente en categorías

positivas, negativas o neutrales. Las soluciones más avanzadas utilizan modelos de aprendizaje automático supervisado entrenados con ejemplos anotados manualmente.

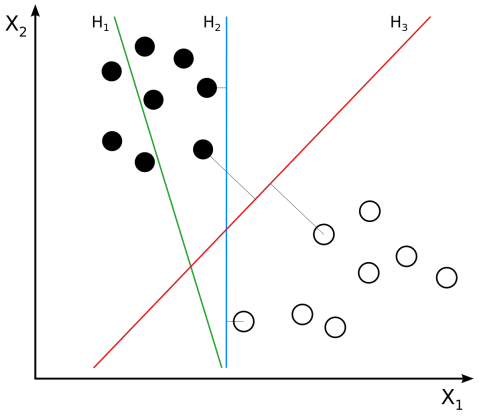
En este tipo de clasificación, es habitual emplear el aprendizaje supervisado, siendo las Máquinas de Vectores de Soporte (SVM) una opción popular. El objetivo de las SVM es encontrar un hiperplano que separe las clases con el margen máximo, logrando la mejor separación entre las clases positivas, negativas y neutrales [Eisenstein, 2018].

- Aplicación de técnicas de **PLN** para identificar y extraer información subjetiva de conjuntos de datos textuales.
- Clasificación automática de frases en las categorías **positiva**, **negativa** o **neutral**.

- Las soluciones más avanzadas emplean modelos de aprendizaje automático **supervisado**, entrenados con ejemplos **anotados manualmente** [Mohammad et al., 2013].



- Idea: Encontrar un hiperplano que separe las clases con el margen máximo (mayor separación).



- H_3 separa las clases con el margen máximo.

1.5.1. Lingüística y Procesamiento del Lenguaje Natural (PNL)

El conocimiento de las estructuras lingüísticas es fundamental para el diseño de características y el análisis de errores en el Procesamiento del Lenguaje Natural (PNL). Los enfoques de aprendizaje automático en PNL se basan en características que describen y generalizan las instancias de uso del lenguaje. El conocimiento lingüístico orienta la selección y el diseño de estas características, ayudando al algoritmo de aprendizaje automático a encontrar correlaciones entre el uso del lenguaje y las etiquetas objetivo [Bender, 2013].

- El conocimiento de las estructuras lingüísticas es importante para el diseño de características y el análisis de errores en PNL [Bender, 2013].
- Los enfoques de aprendizaje automático en PNL requieren características que puedan describir y generalizar el uso del lenguaje.
- El objetivo es guiar al algoritmo de aprendizaje automático para encontrar correlaciones entre el uso del lenguaje y el conjunto de etiquetas objetivo.
- El conocimiento sobre las estructuras lingüísticas puede influir en el diseño de características para los enfoques de aprendizaje automático en PNL.

El PNL plantea diversos desafíos, como los costos de anotación, las variaciones de dominio y la necesidad de actualizaciones continuas. La anotación manual requiere mucho trabajo y tiempo. Las variaciones de dominio implican aprender patrones diferentes para diferentes corpus de texto. Los modelos entrenados en un dominio pueden no funcionar bien en otro. Además, los modelos de PNL pueden volverse obsoletos a medida que el uso del lenguaje evoluciona con el tiempo.

1.6. Desafíos en el Procesamiento del Lenguaje Natural (PNL)

- **Costos de Anotación**: la anotación manual es **laboriosa** y **consume mucho tiempo**.
- **Variaciones de Dominio**: el patrón que queremos aprender puede variar de un corpus a otro (por ejemplo, deportes, política).
- ¡Un modelo entrenado con datos anotados de un dominio no necesariamente funcionará en otro!
- Los modelos entrenados pueden quedar desactualizados con el tiempo (por ejemplo, nuevos hashtags).

Variación de Dominio en el Análisis de Sentimiento

1. Para mí, la cola era bastante **pequeña** y solo tuve que esperar unos 20 minutos, ¡pero valió la pena! :D @raynwise
2. Extraña espacialidad en Stuttgart. La habitación del hotel es tan **pequeña** que apenas puedo moverme, pero los alrededores son inhumanamente vastos y largos bajo construcción.

Superando los costos de anotación de datos Supervisión Distant:

- Etiquetar automáticamente datos no etiquetados (**API de Twitter**) utilizando un método heurístico.
- **Enfoque de Anotación de Emoticonos (EAA)**: los tweets con emoticonos positivos **:)** o negativos **:(** se etiquetan según la polaridad indicada por el emoticono [Read, 2005].
- El emoticono se **elimina** del contenido.
- Este enfoque también se ha ampliado utilizando hashtags como #anger y emojis.
- No es trivial encontrar técnicas de supervisión distante para todo tipo de problemas de PNL.

Crowdsourcing

- Confiar en servicios como **Amazon Mechanical Turk** o **Crowdflower** para solicitar a la **multitud** que anote datos.
- Esto puede resultar costoso.
- Es difícil garantizar la calidad de las anotaciones.

1.7. Estudio de caso: Clasificación de sentimientos en tweets

- En 2013, el taller de Evaluación Semántica (SemEval) organizó la tarea de .^nálisis de sentimientos en Twitter"[Nakov et al., 2013].
- La tarea se dividió en dos sub-tareas: el nivel de expresión y el nivel del mensaje.
- Nivel de expresión: se centró en determinar la polaridad del sentimiento de un mensaje según una entidad marcada dentro de su contenido.
- Nivel del mensaje: se debía determinar la polaridad según el mensaje en general.
- Los organizadores lanzaron conjuntos de datos de entrenamiento y prueba para ambas tareas [Nakov et al., 2013].

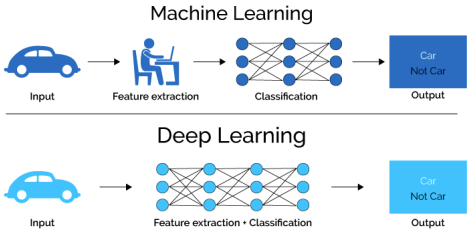
El sistema NRC

- El equipo que logró el mejor rendimiento en ambas tareas, entre 44 equipos, fue el equipo *NRC-Canada* [Mohammad et al., 2013].
- El equipo propuso un enfoque supervisado utilizando un clasificador SVM lineal con las siguientes características hechas a mano para representar los tweets:

  1. N-gramas de palabras.
  2. N-gramas de caracteres.
  3. Etiquetas de partes del discurso.
  4. Agrupaciones de palabras entrenadas con el método de agrupamiento de Brown [Brown et al., 1992].
  5. El número de palabras alargadas (palabras con un carácter repetido más de dos veces).
  6. El número de palabras con todas las letras en mayúscula.
  7. La presencia de emoticonos positivos o negativos.
  8. El número de negaciones individuales.
  9. El número de secuencias contiguas de puntos, signos de interrogación y signos de exclamación.
  10. Características derivadas de lexicones de polaridad [Mohammad et al., 2013]. Dos de estos lexicones se generaron utilizando el método PMI a partir de tweets anotados con hashtags y emoticonos.

1.8. Ingeniería de características y Aprendizaje Profundo

- Hasta 2014, la mayoría de los sistemas de PNL de última generación se basaban en ingeniería de características + modelos de aprendizaje automático superficiales (por ejemplo, SVM, HMM).
- Diseñar las características de un sistema de PNL ganador requiere mucho conocimiento específico del dominio.
- El sistema NRC se construyó antes de que el aprendizaje profundo se hiciera popular en PNL.
- Por otro lado, los sistemas de Aprendizaje Profundo se basan en redes neuronales para aprender automáticamente buenas representaciones.



Ingeniería de características y Aprendizaje Profundo

- El Aprendizaje Profundo proporciona resultados de última generación en la mayoría de las tareas de PNL.
- Grandes cantidades de datos de entrenamiento y máquinas GPU multicore más rápidas son clave en el éxito del aprendizaje profundo.
- Las **redes neuronales** y las **incrustaciones de palabras** desempeñan un papel fundamental en los modelos modernos de PNL.

Aprendizaje Profundo y Conceptos Lingüísticos

- Si los modelos de aprendizaje profundo pueden aprender representaciones automáticamente, ¿siguen siendo útiles los conceptos lingüísticos (por ejemplo, sintaxis, morfología)?
- Algunos defensores del aprendizaje profundo argumentan que estas propiedades lingüísticas inferidas y diseñadas manualmente no son necesarias, y que la red neuronal aprenderá estas representaciones intermedias (o equivalentes o mejores) por sí misma [Goldberg, 2016].
- Aún no hay un consenso definitivo al respecto.

- Goldberg cree que muchos de estos conceptos lingüísticos pueden ser inferidos por la red por sí misma si se le proporciona suficiente cantidad de datos.

- Sin embargo, en muchos otros casos no disponemos de suficientes datos de entrenamiento para la tarea que nos interesa, y en estos casos proporcionar a la red los conceptos generales más explícitos puede ser muy valioso.

1.9. Historia

Los orígenes de PLN se remontan a los años 50 con el famoso test de Alan Turing: una máquina será considerada inteligente cuando sea capaz de conversar con una persona sin que esta pueda determinar si está hablando con una máquina o un ser humano. A lo largo de su historia la disciplina ha tenido tres grandes períodos: 1) el racionalismo, 2) el empirismo, y 3) el aprendizaje profundo [Deng y Liu, 2018] que describimos a continuación.

El racionalismo abarca desde 1950 a 1990, donde las soluciones consistían en diseñar reglas manuales para incorporar mecanismos de conocimiento y razonamiento. Un ejemplo emblemático es el agente de conversación (o chatbot) ELIZA desarrollado por Joseph Weizenbaum que simulaba un psicoterapeuta rogeriano. Luego, a partir de la década de los 90s, el diseño de métodos estadísticos y de aprendizaje automático construidos sobre corpus llevan a PLN hacia un enfoque empirista. Las reglas ya no se construyen sino que se "aprenden" a partir de datos etiquetados. Algunos modelos representativos de esta época son los filtros de spam basados en modelos lineales, las cadenas de Markov ocultas para la extracción de categorías sintácticas y los modelos probabilísticos de IBM para la traducción automática. Estos modelos se caracterizaban por ser poco profundos en su estructura de parámetros y por depender de características manualmente diseñadas para representar la entrada.

A partir del año 2010, las redes neuronales artificiales, que son una familia de modelos de aprendizaje automático, comienzan a mostrar resultados muy superiores en varias tareas emblemáticas de PLN [Collobert et al., 2011]. La idea de estos modelos es representar la entrada (el texto) con una jerarquía de parámetros (o capas) que permiten encontrar representaciones idóneas para la tarea en cuestión, proceso al cual se refiere como "aprendizaje profundo". Estos modelos se caracterizan por tener muchos más parámetros que los modelos anteriores (superando la barrera del millón en algunos casos) y requerir grandes volúmenes de datos para su entrenamiento. Una gracia de estos modelos es que pueden ser pre-entrenados con texto no etiquetado como libros, Wikipedia, texto de redes sociales y de la Web para encontrar representaciones iniciales de palabras y oraciones (a lo que conocemos como word embeddings), las cuales pueden ser posteriormente adaptadas para la tarea objetivo donde sí se tienen datos etiquetados (Proceso conocido como transfer learning). Aquí destacamos modelos como Word2Vec [Mikolov 2013], BERT [Devlin 2018] y

GPT-3 [Brown 2020].

Este tipo de modelos ha ido perfeccionándose en los últimos años, llegando a obtener resultados cada vez mejores para casi todos los problemas del área [NLPProgress]. Sin embargo, este progreso no ha sido libre de controversias. El aumento exponencial en la cantidad de parámetros de cada nuevo modelo respecto a su predecesor, hace que los recursos computacionales y energéticos necesarios para construirlos sólo estén al alcance de unos pocos. Además, varios estudios han mostrado que estos modelos aprenden y reproducen los sesgos y prejuicios (ej: género, religión, racial) presentes en los textos a partir de los cuales se entrenan. Sin ir más lejos, la investigadora Timmnit Gebru fue despedida de Google cuando se le negó el permiso para publicar un artículo que ponía de manifiesto estos problemas [Bender 2021].

El progreso de la PNL se puede dividir en tres oleadas principales: 1) racionalismo, 2) empirismo y 3) aprendizaje profundo [Deng and Liu, 2018].

1950 - 1990 Racionalismo: se enfocaba en diseñar reglas hechas a mano para incorporar conocimiento y mecanismos de razonamiento en sistemas de PNL inteligentes (por ejemplo, ELIZA para simular a un psicoterapeuta Rogeriano, MARGIE para estructurar información del mundo real en ontologías de conceptos).

1991 - 2009 Empirismo: se caracteriza por la explotación de corpora de datos y modelos de aprendizaje automático y estadísticos (superficiales) (por ejemplo, Naive Bayes, HMMs, modelos de traducción IBM).

2010 - Aprendizaje Profundo: la ingeniería de características (considerada como un cuello de botella) se reemplaza con el aprendizaje de representaciones y/o redes neuronales profundas (por ejemplo, `https://www.deepl.com/translator`). Un artículo muy influyente en esta revolución: [Collobert et al., 2011].

1.10. Conclusiones

En este capítulo, hemos explorado el desafío de entender y producir lenguaje utilizando computadoras. El aprendizaje automático supervisado es una de las principales técnicas utilizadas para abordar este desafío. Además, hemos discutido las propiedades desafiantes del lenguaje, como la discreción, la composicionalidad y la dispersión. Estos aspectos nos muestran la complejidad inherente al procesamiento del lenguaje natural y nos desafían a encontrar soluciones efectivas.

[1] Las fechas son aproximadas.

Capítulo 2

Modelo de Espacio Vectorial y Recuperación de Información

- ¿Cómo recupera un motor de búsqueda, como Duckduckgo o Google, los documentos relevantes a partir de una consulta dada?
- ¿Cómo puede una empresa procesar las reclamaciones dejadas por sus usuarios en sus portales web?

Estos problemas se estudian en los siguientes campos:

- *Recuperación de Información*: ciencia de buscar información en colecciones de documentos.
- *Minería de Texto*: extracción automática de conocimiento a partir de texto.

¡Ambos están estrechamente relacionados con el Procesamiento del Lenguaje Natural (NLP, por sus siglas en inglés)! (las fronteras entre estos campos no están claras).

2.1. Tokens y Tipos

Tokenización: la tarea de dividir una oración o documento en fragmentos llamados *tokens*.
Se pueden emplear transformaciones adicionales, como la eliminación de caracteres especiales (por ejemplo, puntuación), minúsculas, etc. [Manning et al., 2008].

Ejemplo Entrada: Me gustan los lenguajes humanos y los lenguajes de programación.
Tokens: [Me] [gustan] [los] [lenguajes] [humanos] [y] [los] [lenguajes] [de] [programación]

Tipos

- Un *tipo* es una clase de *token* que contiene una única secuencia de caracteres.
- Se obtienen identificando los tokens únicos dentro del documento.

Tipos para la oración anterior: [Me] [gustan] [los] [lenguajes] [humanos] [y] [de] [programación] El token *lenguajes* se repitió en la oración.

Extracción de Vocabulario

- Un *término* es un *tipo* normalizado.
- La normalización es el proceso de crear clases de equivalencia de diferentes *tipos*. Esto quedará claro en las siguientes diapositivas.
- El vocabulario V es el conjunto de términos (tokens únicos normalizados) dentro de una colección de documentos o corpus D.

Eliminación de stopwords

- Con el fin de reducir el tamaño del vocabulario y eliminar términos que no aportan mucha información, se eliminan los términos que ocurren con alta frecuencia en el corpus.
- Estos términos se llaman *stopwords* e incluyen artículos, pronombres, preposiciones y conjunciones. Ejemplo: [un, una, y, cualquier, tiene, hacer, no, hizo, el, en].

¡La eliminación de stopwords puede ser inconveniente en muchas tareas de procesamiento del lenguaje natural!

Ejemplo: No me gusta la pizza => pizza (se eliminaron "no", "me" "gusta")

Stemming Es un proceso de normalización de términos en el cual los términos se transforman a su raíz con el objetivo de reducir el tamaño del vocabulario. Se lleva a cabo aplicando reglas de reducción de palabras. Ejemplo: Algoritmo de Porter.

(F)

Rule			Example		
SSES	→	SS	caresses	→	caress
IES	→	I	ponies	→	poni
SS	→	SS	caress	→	caress
S	→		cats	→	cat

Ejemplo: d = Me gustan los lenguajes humanos y los lenguajes de programación => Me gustan los lenguaj y los program lenguaj[1]

El vocabulario del documento d después de eliminar stopwords y realizar stemming:

[1] `http://9ol.es/porter_js_demo.html`

termId	value
t1	human
t2	languag
t3	program

Lematización

- Otra estrategia de normalización de términos.
- También transforma las palabras en sus raíces.
- Realiza un análisis morfológico utilizando diccionarios de referencia (tablas de búsqueda) para crear clases de equivalencia entre *tipos*.
- Por ejemplo, para el token *estudios*, una regla de stemming devolvería el término *estudi*, mientras que a través de la lematización obtendríamos el término *study*[2].

2.1.1. Ley de Zipf

- La Ley de Zipf, propuesta por *George Kingsley Zipf* en [Zipf, 1935], es una ley empírica sobre la frecuencia de los términos dentro de una colección de documentos (**corpus**).
- Establece que la frecuencia f de un término en un corpus es inversamente proporcional a su posición r en una tabla de frecuencia ordenada:

$$f = \frac{cf}{r^{\beta}} \tag{2.1}$$

- Donde cf es una constante dependiente de la colección y $\beta > 0$ es un factor de decaimiento.
- Si $\beta = 1$, entonces f sigue exactamente la Ley de Zipf; de lo contrario, sigue una distribución similar a la de Zipf.
- La ley se relaciona con el principio del mínimo esfuerzo. A menudo utilizamos pocas palabras para expresar ideas.
- La Ley de Zipf es un tipo de distribución de ley de potencia (distribuciones de cola larga).
- Si trazamos un gráfico de *log-log*, obtenemos una línea recta con una pendiente de $-\beta$.
- Enumerar las palabras más frecuentes de un corpus se puede utilizar para construir una lista de *stopwords*.

[2] https://blog.bitext.com/what-is-the-difference-between-stemming-and-lemmatization/

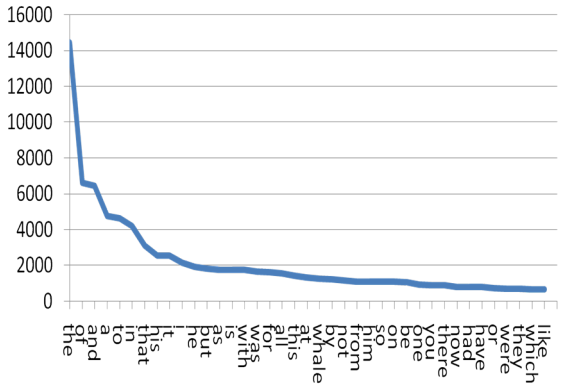


Figura 2.1: Ley de Zipf

2.1.2. Listas de publicaciones y el índice invertido

Sea D una colección de documentos y V el vocabulario de todos los términos extraídos de la colección:

- La lista de publicaciones de un término es la lista de todos los documentos donde el término aparece al menos una vez. Los documentos se identifican por sus identificadores.
- Un índice invertido es una estructura de datos tipo diccionario que mapea los términos $t_i \in V$ con sus listas de publicaciones correspondientes.

$$< \text{término} > \rightarrow < \text{idDocumento} >^*$$

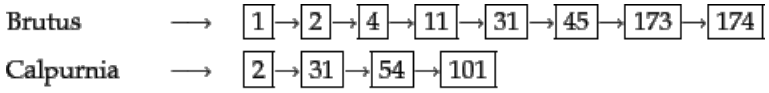


Figura 2.2: Índice invertido

2.1.3. Motores de búsqueda web

Un motor de búsqueda es un sistema de recuperación de información diseñado para buscar información en la web (satisfacer necesidades de información) [Manning et al., 2008]. Sus componentes básicos son:

- Rastreador: un robot que navega por la web según una estrategia definida. Por lo general, comienza navegando por un conjunto de sitios web iniciales y continúa navegando a través de sus enlaces.

- Indexador: se encarga de mantener un índice invertido con el contenido de las páginas recorridas por el rastreador.
- Procesador de consultas: se encarga de procesar las consultas de los usuarios y buscar en el índice los documentos más relevantes para una consulta.
- Función de clasificación: la función utilizada por el procesador de consultas para clasificar los documentos indexados en la colección por relevancia según una consulta.
- Interfaz de usuario: recibe la consulta como entrada y devuelve los documentos clasificados por relevancia.

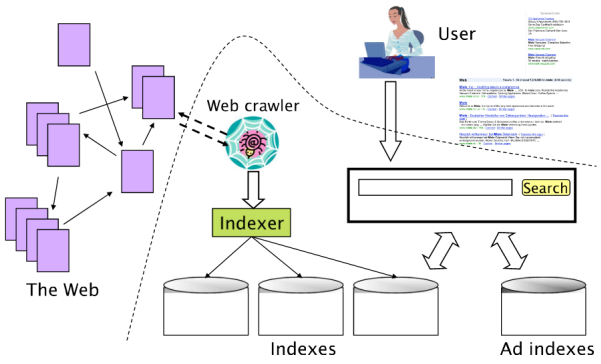


Figura 2.3: Los diversos componentes de un motor de búsqueda web [Manning et al., 2008].

2.2. El modelo de espacio vectorial

- Para clasificar consultas o medir la similitud entre dos documentos, necesitamos una métrica de similitud.
- Los documentos pueden ser *representados* como vectores de términos, donde cada término es una dimensión del vector [Salton et al., 1975].
- Documentos con diferentes palabras y longitudes residirán en el mismo espacio vectorial.
- Este tipo de representaciones se llaman *Bolsa de Palabras* (Bag of Words).
- En las representaciones de bolsa de palabras, se pierde el orden de las palabras y la estructura lingüística de una oración.

- El valor de cada dimensión es un peso que representa la relevancia del término t_i en el documento d.

$$d_j \rightarrow \overrightarrow{d_j} = (w(t_1, d_j), ..., w(t_{|V|}, d_j)) \tag{2.2}$$

- ¿Cómo podemos modelar la información que aporta un término a un documento?

Frecuencia de Término - Frecuencia Inversa de Documento

- Sea $tf_{i,j}$ la frecuencia del término t_i en el documento d_j.
- Un término que ocurre 10 veces debería proporcionar más información que uno que ocurre solo una vez.
- ¿Qué ocurre cuando tenemos documentos que son mucho más largos que otros?
- Podemos normalizar dividiendo por la frecuencia máxima del término en el documento.

$$ntf_{i,j} = \frac{tf_{i,j}}{\text{máx}_i(tf_{i,j})}$$

- ¿Un término que ocurre en muy pocos documentos proporciona más o menos información que uno que ocurre varias veces?
- Por ejemplo, el documento *El respetado alcalde de Pelotillehue*. El término *Pelotillehue* ocurre en menos documentos que el término *alcalde*, por lo que debería ser más descriptivo.
- Sea N el número de documentos en la colección y n_i el número de documentos que contienen el término t_i, definimos la frecuencia inversa de documento (idf) de t_i de la siguiente manera:

$$idf_{t_i} = \log_{10}\left(\frac{N}{n_i}\right)$$

- Un término que aparece en todos los documentos tendría $idf = 0$, y uno que aparece en el $10\,\%$ de los documentos tendría $idf = 1$.
- El modelo de puntuación tf-idf combina las puntuaciones de tf e idf, y resulta en los siguientes pesos w para un término en un documento:

$$w(t_i, d_j) = tf_i \times \log_{10}\left(\frac{N}{n_i}\right)$$

- Las consultas de los motores de búsqueda también pueden ser modeladas como vectores. Sin embargo, en promedio, las consultas suelen tener entre 2 y 3 términos. Para evitar tener demasiadas dimensiones nulas, los vectores de consulta pueden suavizarse de la siguiente manera:

$$w(t_i, d_j) = (0{,}5 + 0{,}5 \times tf_{i,j}) \log_{10}\left(\frac{N}{n_i}\right)$$

2.2.1. Similitud entre vectores

- Representar consultas y documentos como vectores permite calcular su similitud.
- Un enfoque podría ser utilizar la distancia euclidiana.
- El enfoque común es calcular el coseno del ángulo entre los dos vectores.
- Si ambos documentos son iguales, el ángulo sería 0 y su coseno sería 1. Por otro lado, si son ortogonales, el coseno es 0.
- La similitud del coseno se calcula de la siguiente manera:

$$\text{similitud del coseno}(\vec{d1}, \vec{d2}) = \frac{\vec{d1} \cdot \vec{d2}}{|\vec{d1}| \times |\vec{d2}|} = \frac{\sum_{i=1}^{|V|}(w(t_i, d_1) \times w(t_i, d_2))}{\sqrt{\sum_{i=1}^{|V|} w(t_i, d_1)^2} \times \sqrt{\sum_{i=1}^{|V|} w(t_i, d_2)^2}}$$

- Esto se llama incorrectamente "distancia del coseno". En realidad, es una métrica de similitud.
- Observa que la similitud del coseno normaliza los vectores por su norma euclidiana $||\vec{d}||_2$.

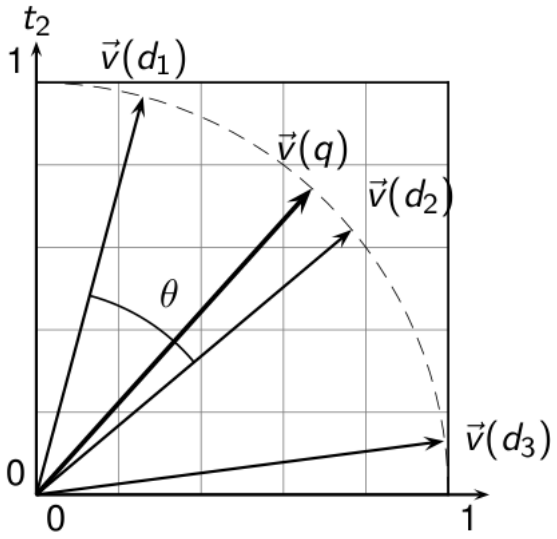


Figura 2.4: Similitud del coseno.

Ejercicio

- Supongamos que tenemos 3 documentos formados a partir de las siguientes secuencias de términos: $d_1 \rightarrow t_4t_3t_1t_4$ $d_2 \rightarrow t_5t_4t_2t_3t_5$ $d_3 \rightarrow t_2t_1t_4t_4$
- Construye una matriz término-documento de dimensiones 5×3 utilizando pesos simples de tf-idf (sin normalización).
- Recomendamos que primero construyas una lista con el número de documentos en los que aparece cada término (útil para calcular los valores de idf).
- Luego, calcula los valores de idf para cada término.
- Rellena las celdas de la matriz con los valores de tf-idf.
- ¿Cuál es el documento más cercano a d_1?

	d1	d2	d3
t1	0.176	0.000	0.176
t2	0.000	0.176	0.176
t3	0.176	0.176	0.000
t4	0.000	0.000	0.000
t5	0.000	0.954	0.000

Cuadro 2.1: Matriz tf-idf

2.3. Agrupamiento de Documentos

- ¿Cómo podemos agrupar documentos que son similares entre sí?
- El agrupamiento es el proceso de agrupar documentos que son similares entre sí.
- Cada grupo de documentos se llama *cluster* o grupo.
- En el agrupamiento, intentamos identificar grupos de documentos en los que la similitud entre documentos en el mismo grupo se maximiza y la similitud de documentos en diferentes grupos se minimiza.
- El agrupamiento de documentos permite identificar temas en un corpus y reducir el espacio de búsqueda en un motor de búsqueda, es decir, el índice invertido se organiza según los grupos.
- K-means es un algoritmo de agrupamiento simple que recibe el número de grupos k como parámetro.

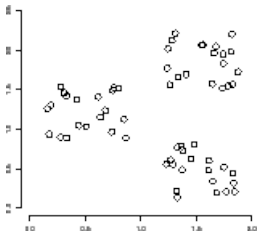


Figura 2.5: Conjunto de documentos donde los grupos se pueden identificar claramente.

- El algoritmo se basa en la idea de *centroide*, que es el vector promedio de los documentos que pertenecen al mismo grupo.
- Sea S un conjunto de vectores bidimensionales $3, 6, 1, 2, 5, 1$, el centroide de S es $(3 + 1 + 5)/3, (6 + 2 + 1)/3 = 3, 3$.

2.3.1. K-Means

1. Comenzamos con k centroides aleatorios.
2. Calculamos la similitud entre cada documento y cada centroide.
3. Asignamos cada documento a su centroide más cercano formando un grupo.
4. Se recalculan los centroides de acuerdo a los documentos asignados a ellos.
5. Este proceso se repite hasta la convergencia.

2.4. Conclusiones y Conceptos Adicionales

- Representar documentos como vectores es fundamental para calcular similitudes entre pares de documentos.
- Los vectores de "bag of wordsçarecen de estructura lingüística.
- Los vectores de "bag of words"son de alta dimensionalidad y dispersos.
- Los n-gramas de palabras pueden ayudar a capturar expresiones de múltiples palabras (por ejemplo, New York => new_york)
- Los sistemas modernos de recuperación de información van más allá de la similitud de vectores (PageRank, Retroalimentación de relevancia, Minería de registros de consultas, Grafo de conocimiento de Google, Aprendizaje automático).

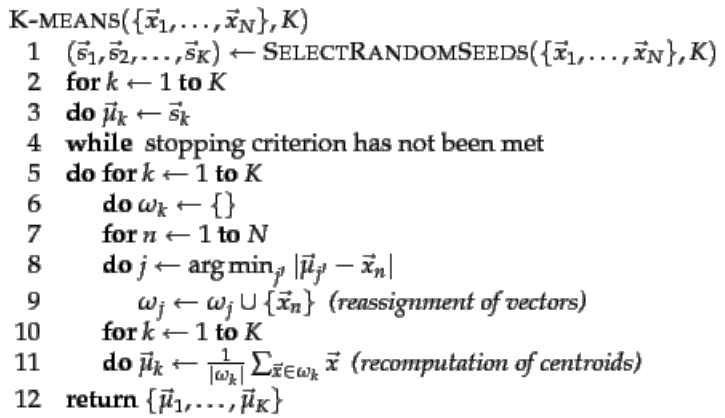


Figura 2.6: Algoritmo K-means

- La recuperación de información y la minería de textos se preocupan menos por la estructura lingüística y más por producir algoritmos rápidos y escalables [Eisenstein, 2018].

Capítulo 3

Modelos de Lenguaje Probabilísticos

3.1. The Language Modeling Problem

- We have some (finite) vocabulary, say $\mathcal{V} = \{$the, a, man, telescope, Beckham, two, . . .$\}$
- We have an (infinite) set of strings, $\mathcal{V}^*$.
- For example:
  - the STOP
  - a STOP
  - the fan STOP
  - the fan saw Beckham STOP
  - the fan saw saw STOP
  - the fan saw Beckham play for Real Madrid STOP
- Where STOP is a special symbol indicating the end of a sentence.
- We have a training sample of example sentences in English.
- We need to "learn" a probability distribution p.
- p is a function that satisfies:

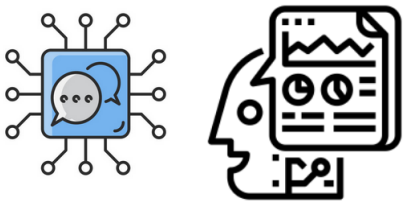
$$\sum_{x \in V^*} p(x) = 1$$

$$p(x) \geq 0 \quad \text{for all } x \in V^*$$

- Examples of probabilities assigned to sentences:

$$
\begin{aligned}
p(\text{the STOP}) &= 10^{-12} \\
p(\text{the fan STOP}) &= 10^{-8} \\
p(\text{the fan saw Beckham STOP}) &= 2 \times 10^{-8} \\
p(\text{the fan saw saw STOP}) &= 10^{-15} \\
&\dots \\
p(\text{the fan saw Beckham play for Real Madrid STOP}) &= 2 \times 10^{-9}
\end{aligned}
$$

- Idea 1: The model assigns a higher probability to fluent sentences (those that make sense and are grammatically correct).
- Idea 2: Estimating this probability function from text (corpus).
- The language model helps text generation models distinguish between good and bad sentences.



3.1.1. Why would we want to do this?

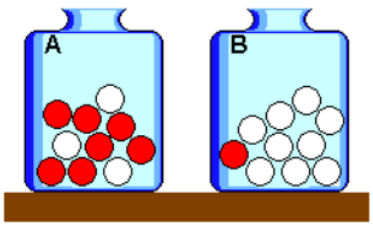
- Speech recognition was the original motivation.
- Consider the sentences: 1) recognize speech and 2) wreck a nice beach.
- These two sentences sound very similar when pronounced, making it challenging for automatic speech recognition systems to accurately transcribe them.
- When the speech recognition system analyzes the audio input and tries to transcribe it, it takes into account the language model probabilities to determine the most likely interpretation.
- The language model would favor p(recognize speech) over p(wreck a nice beach).
- This is because the former is a more common sentence and should occur more frequently in the training corpus.

item By incorporating language models, speech recognition systems can improve accuracy by selecting the sentence that aligns better with linguistic patterns and context, even when faced with similar-sounding alternatives.

- Related problems are optical character recognition, handwriting recognition.
- Acutally, Language Models are useful in any NLP tasks involving the generation of language (e.g., machine translation, summarization, chatbots).
- The estimation techniques developed for this problem will be VERY useful for other problems in NLP.

3.1.2. Language Models are Generative

- Language models can generate sentences by sequentially sampling from probabilities.
- This is analogous to drawing balls (words) from an urn where their sizes are proportional to their relative frequencies.
- Alternatively, one could always draw the most probable word, which is equivalent to predicting the next word.



3.1.3. A Naive Method

- A very naive method for estimating the probability of a sentence is to count the occurrences of the sentence in the training data and divide it by the total number of training sentences (N) to estimate the probability.
- We have N training sentences.
- For any sentence $x_1, x_2, \ldots, x_n$, $c(x_1, x_2, \ldots, x_n)$ is the number of times the sentence is seen in our training data.

- A naive estimate:

$$p(x1, x2, \ldots, xn) = \frac{c(x_1, x_2 \ldots, x_n)}{N}$$

- Problem: As the number of possible sentences grows exponentially with sentence length and vocabulary size, it becomes increasingly unlikely for a specific sentence to appear in the training data.

- Consequently, many sentences will have a probability of zero according to the naive model, leading to poor generalization.

3.2. Markov Processes

- Consider a sequence of random variables $X_1, X_2, \ldots, X_n$.

- Each random variable can take any value in a finite set V.

- For now, we assume the length n is fixed (e.g., $n = 100$).

- Our goal: model $P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$

First-Order Markov Processes

$$\begin{aligned} P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) &= P(X_1 = x_1) \prod_{i=2}^{n} P(X_i = x_i | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}) \\ &= P(X_1 = x_1) \prod_{i=2}^{n} P(X_i = x_i | X_{i-1} = x_{i-1}) \end{aligned}$$

The first-order Markov assumption: For any $i \in \{2, \ldots, n\}$ and any $x_1, \ldots, x_i$,

$$P(X_i = x_i | X_1 = x_1, \ldots, X_{i-1} = x_{i-1}) = P(X_i = x_i | X_{i-1} = x_{i-1})$$

Second-Order Markov Processes

$$\begin{aligned} &P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = \\ &P(X_1 = x_1) \cdot P(X_2 = x_2 | X_1 = x_1) \cdot \prod_{i=3}^{n} P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}) \\ &= \prod_{i=1}^{n} P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}) \end{aligned}$$

(For convenience, we assume $x_0 = x_{-1} = *$, where $*$ is a special "start"symbol.)

3.2.1. Modeling Variable Length Sequences

- We would like the length of the sequence, n, to also be a random variable.
- A simple solution: always define $X_n = \text{STOP}$, where STOP is a special symbol.
- Then use a Markov process as before:

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = \prod_{i=1}^{n} P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})$$

- (For convenience, we assume $x_0 = x_{-1} = *$, where $*$ is a special "start"symbol.)

3.3. Trigram Language Models

- A trigram language model consists of:
  1. A finite set V
  2. A parameter $q(w|u, v)$ for each trigram u, v, w such that $w \in V \cup \{\text{STOP}\}$, and $u, v \in V \cup \{*\}$
- For any sentence $x_1 \ldots x_n$ where $x_i \in V$ for $i = 1 \ldots (n-1)$, and $x_n = \text{STOP}$, the probability of the sentence under the trigram language model is:

$$p(x_1 \ldots x_n) = \prod_{i=1}^{n} q(x_i | x_{i-2}, x_{i-1})$$

- We define $x_0 = x_{-1} = *$ for convenience.

An Example For the sentence `the dog barks STOP`, we would have:

$$p(\text{the dog barks STOP}) = q(\text{the}|*, *) \times q(\text{dog}|*, \text{the}) \times q(\text{barks}|\text{the, dog}) \times q(\text{STOP}|\text{dog, barks})$$

3.3.1. The Trigram Estimation Problem

Remaining estimation problem:

$$q(w_i | w_{i-2}, w_{i-1})$$

For example:

$$q(\text{laughs}|\text{the, dog})$$

A natural estimate (the "maximum likelihood estimate"):

$$q(w_i | w_{i-2}, w_{i-1}) = \frac{\text{Count}(w_{i-2}, w_{i-1}, w_i)}{\text{Count}(w_{i-2}, w_{i-1})}$$

For instance,

$$q(\text{laughs}|\text{the, dog}) = \frac{\text{Count(the, dog, laughs)}}{\text{Count(the, dog)}}$$

Sparse Data Problems A natural estimate (the "maximum likelihood estimate"):

$$q(w_i | w_{i-2}, w_{i-1}) = \frac{\text{Count}(w_{i-2}, w_{i-1}, w_i)}{\text{Count}(w_{i-2}, w_{i-1})}$$

$$q(\text{laughs}|\text{the, dog}) = \frac{\text{Count(the, dog, laughs)}}{\text{Count(the, dog)}}$$

- Say our vocabulary size is $N = |V|$, then there are N^3 parameters in the model.
- For example, $N = 20,000 \Rightarrow 20,000^3 = 8 \times 10^{12}$ parameters.

3.4. Evaluating a Language Model: Perplexity

- We have some test data, m sentences: $s_1, s_2, s_3, ..., s_m$
- We could look at the probability under our model $\prod_{i=1}^{m} p(s_i)$. Or more conveniently, the log probability:

$$\log\left(\prod_{i=1}^{m} p(s_i)\right) = \sum_{i=1}^{m} \log p(s_i)$$

- In fact, the usual evaluation measure is perplexity:

$$\text{Perplexity} = 2^{-l} \quad \text{where} \quad l = \frac{1}{M}\sum_{i=1}^{m} \log p(s_i)$$

- M is the total number of words in the test data

Some Intuition about Perplexity

- Say we have a vocabulary V, and $N = |V| + 1$, and a model that predicts:

$$q(w|u, v) = \frac{1}{N} \quad \text{for all } w \in V \cup \{\text{STOP}\}, \text{ for all } u, v \in V \cup \{*\}$$

- It's easy to calculate the perplexity in this case:

$$\text{Perplexity} = 2^{-l} \quad \text{where} \quad l = \log\frac{1}{N} \Rightarrow \text{Perplexity} = N$$

- Perplexity can be seen as a measure of the effective "branching factor".
- **Proof:** Let's asume we have m sentences of length n in the corpus, and M the amount of tokens in the corpus, $M = m \cdot n$.
- Let's consider the log (base 2) probability of a sentence $s = w_1 w_2 \dots w_n$ under the model:

$$\log p(s) = \log \prod_{i=1}^{n} q(w_i|w_{i-2}, w_{i-1}) = \sum_{i=1}^{n} \log q(w_i|w_{i-2}, w_{i-1})$$

- Since each $q(w_i|w_{i-2}, w_{i-1})$ is equal to $\frac{1}{N}$, we have:

$$\log p(s) = \sum_{i=1}^{n} \log\frac{1}{N} = n \cdot \log\frac{1}{N} = -n \cdot \log N$$

$$l = \frac{1}{M}\sum_{i=1}^{m} \log p(s_i) = \frac{1}{M}\sum_{i=1}^{m} -n \cdot \log N = \frac{1}{M} \cdot -m \cdot n \cdot \log N = -\log N$$

- Therefore, the perplexity is given by:

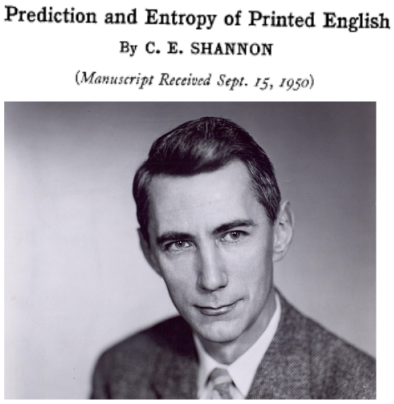
$$\text{Perplexity} = 2^{-l} = 2^{-(-\log N)} = N$$

Typical Values of Perplexity

- Results from Goodman (."A bit of progress in language modeling"), where $|V| = 50,000$ [Goodman, 2001].
- A trigram model: $p(x_1, \dots, x_n) = \prod_{i=1}^{n} q(x_i|x_{i-2}, x_{i-1})$
  Perplexity = 74
- A bigram model: $p(x_1, \dots, x_n) = \prod_{i=1}^{n} q(x_i|x_{i-1})$
  Perplexity = 137
- A unigram model: $p(x_1, \dots, x_n) = \prod_{i=1}^{n} q(x_i)$
  Perplexity = 955

3.5. Some History

- Shannon conducted experiments on the entropy of English, specifically investigating how well people perform in the perplexity game.
- Reference: C. Shannon. "Prediction and entropy of printed English." *Bell Systems Technical Journal*, 30:50–64, 1951. [Shannon, 1951]


Prediction and Entropy of Printed English

By C. E. SHANNON

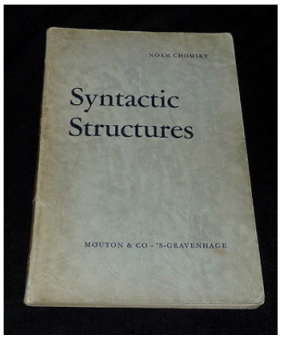
(Manuscript Received Sept. 15, 1950)

- Chomsky, in his book *Syntactic Structures* (1957), made several important points regarding grammar. [Chomsky, 2009]
- According to Chomsky, the notion of "grammaticalçannot be equated with "meaningful.ºr "significantïn a semantic sense.
- He illustrated this with two nonsensical sentences:
  - (1) Colorless green ideas sleep furiously.
  - (2) Furiously sleep ideas green colorless.
- While both sentences lack meaning, Chomsky argued that only the first one is considered grammatical by English speakers.
- Chomsky also emphasized that grammaticality in English cannot be determined solely based on statistical approximations.
- Even though neither sentence (1) nor (2) has likely occurred in English discourse, a statistical model would consider them equally remote"from English.
- However, sentence (1) is grammatical, while sentence (2) is not, highlighting the limitations of statistical approaches in capturing grammaticality.

3.6. The Bias-Variance Trade-Off

- Trigram maximum-likelihood estimate:

$$q_{\text{ML}}(w_i|w_{i-2}, w_{i-1}) = \frac{\text{Count}(w_{i-2}, w_{i-1}, w_i)}{\text{Count}(w_{i-2}, w_{i-1})}$$



- Bigram maximum-likelihood estimate:

$$q_{\text{ML}}(w_i|w_{i-1}) = \frac{\text{Count}(w_{i-1}, w_i)}{\text{Count}(w_{i-1})}$$

- Unigram maximum-likelihood estimate:

$$q_{\text{ML}}(w_i) = \frac{\text{Count}(w_i)}{\text{Count}()}$$

3.7. Linear Interpolation

- Take our estimate $q(w_i|w_{i-2}, w_{i-1})$ to be

$$q(w_i|w_{i-2}, w_{i-1}) = \lambda_1 \cdot q_{\text{ML}}(w_i|w_{i-2}, w_{i-1}) + \lambda_2 \cdot q_{\text{ML}}(w_i|w_{i-1}) + \lambda_3 \cdot q_{\text{ML}}(w_i)$$

  where $\lambda_1 + \lambda_2 + \lambda_3 = 1$, and $\lambda_i \geq 0$ for all i.

- Our estimate correctly defines a distribution (define $V' = V \cup \{\text{STOP}\}$):

$$\begin{aligned}
&\sum_{w \in V'} q(w|u, v) \\
=&\sum_{w \in V'} [\lambda_1 \cdot q_{\text{ML}}(w|u, v) + \lambda_2 \cdot q_{\text{ML}}(w|v) + \lambda_3 \cdot q_{\text{ML}}(w)] \\
=&\lambda_1 \sum_w q_{\text{ML}}(w|u, v) + \lambda_2 \sum_w q_{\text{ML}}(w|v) + \lambda_3 \sum_w q_{\text{ML}}(w) \\
=&\lambda_1 + \lambda_2 + \lambda_3 = 1
\end{aligned}$$

- We can also show that $q(w|u, v) \geq 0$ for all $w \in V'$.

3.8. Estimating λ Values

- Hold out part of the training set as *validation* data.
- Define $c'(w_1, w_2, w_3)$ to be the number of times the trigram (w_1, w_2, w_3) is seen in the validation set.
- Choose $\lambda_1, \lambda_2, \lambda_3$ to maximize:

$$L(\lambda_1, \lambda_2, \lambda_3) = \sum_{w_1, w_2, w_3} c'(w_1, w_2, w_3) \log q(w_3|w_1, w_2)$$

  such that $\lambda_1 + \lambda_2 + \lambda_3 = 1$, and $\lambda_i \geq 0$ for all i, and where

$$q(w_i|w_{i-2}, w_{i-1}) = \lambda_1 \cdot q_{\text{ML}}(w_i|w_{i-2}, w_{i-1}) + \lambda_2 \cdot q_{\text{ML}}(w_i|w_{i-1}) + \lambda_3 \cdot q_{\text{ML}}(w_i)$$

3.9. Discounting Methods

- Consider the following counts and maximum-likelihood estimates:

Sentence	**Count**	$q_{\mathbf{ML}}(w_i \mid w_{i-1})$
the	48	
the, dog	15	$15/48$
the, woman	11	$11/48$
the, man	10	$10/48$
the, park	5	$5/48$
the, job	2	$2/48$
the, telescope	1	$1/48$
the, manual	1	$1/48$
the, afternoon	1	$1/48$
the, country	1	$1/48$
the, street	1	$1/48$

- The maximum-likelihood estimates are high, particularly for low count items.
- Define "discounted" counts as follows:

$$\text{Count}^*(x) = \text{Count}(x) - 0{,}5$$

Sentence	**Count**	**Count*(x)**	$\mathbf{q}_{\text{ML}}(\mathbf{w_i} \mid \mathbf{w_{i-1}})$
the	48		
the, dog	15	14.5	$14{,}5/48$
the, woman	11	10.5	$10{,}5/48$
the, man	10	9.5	$9{,}5/48$
the, park	5	4.5	$4{,}5/48$
the, job	2	1.5	$1{,}5/48$
the, telescope	1	0.5	$0{,}5/48$
the, manual	1	0.5	$0{,}5/48$
the, afternoon	1	0.5	$0{,}5/48$
the, country	1	0.5	$0{,}5/48$
the, street	1	0.5	$0{,}5/48$

- The new estimates are based on the discounted counts.
- We now have some "missing probability mass":

$$\alpha(w_{i-1}) = 1 - \sum_w \frac{\text{Count}^*(w_{i-1}, w)}{\text{Count}(w_{i-1})}$$

- For example, in our case:

$$\alpha(\text{the}) = \frac{10 \times 0{,}5}{48} = \frac{5}{48}$$

3.9.1. Katz Back-Off Models (Bigrams)

- For a bigram model, define two sets:

$$A(w_{i-1}) = \{w : \text{Count}(w_{i-1}, w) > 0\}$$

$$B(w_{i-1}) = \{w : \text{Count}(w_{i-1}, w) = 0\}$$

- A bigram model:

$$q_{\text{BO}}(w_i|w_{i-1}) = \begin{cases} \frac{\text{Count}^*(w_{i-1},w_i)}{\text{Count}(w_{i-1})} & \text{if } w_i \in A(w_{i-1}) \\ \frac{\alpha(w_{i-1})q_{\text{ML}}(w_i)}{\sum_{w\in B(w_{i-1})} q_{\text{ML}}(w)} & \text{if } w_i \in B(w_{i-1}) \end{cases}$$

- Where:

$$\alpha(w_{i-1}) = 1 - \sum_{w\in A(w_{i-1})} \frac{\text{Count}^*(w_{i-1}, w)}{\text{Count}(w_{i-1})}$$

- For a trigram model, first define two sets:

$$A(w_{i-2}, w_{i-1}) = \{w : \text{Count}(w_{i-2}, w_{i-1}, w) > 0\}$$

$$B(w_{i-2}, w_{i-1}) = \{w : \text{Count}(w_{i-2}, w_{i-1}, w) = 0\}$$

- A trigram model is defined in terms of the bigram model:

$$q_{\text{BO}}(w_i|w_{i-2}, w_{i-1}) = \begin{cases} \frac{\text{Count}^*(w_{i-2},w_{i-1},w_i)}{\text{Count}(w_{i-2},w_{i-1})} & \text{if } w_i \in A(w_{i-2}, w_{i-1}) \\ \frac{\alpha(w_{i-2},w_{i-1})q_{\text{BO}}(w_i|w_{i-1})}{\sum_{w\in B(w_{i-2},w_{i-1})} q_{\text{BO}}(w|w_{i-1})} & \text{if } w_i \in B(w_{i-2}, w_{i-1}) \end{cases}$$

- Where:

$$\alpha(w_{i-2}, w_{i-1}) = 1 - \sum_{w\in A(w_{i-2},w_{i-1})} \frac{\text{Count}^*(w_{i-2}, w_{i-1}, w)}{\text{Count}(w_{i-2}, w_{i-1})}$$

3.10. Summary

- Deriving probabilities in probabilistic language models involves three steps:

  1. Expand $p(w_1, w_2, \ldots, w_n)$ using the Chain rule.
  2. Apply Markov Independence Assumptions
     $p(w_i|w_1, w_2, \ldots, w_{i-2}, w_{i-1}) = p(w_i|w_{i-2}, w_{i-1})$.
  3. Smooth the estimates using low order counts.

- Other methods for improving language models include:

  - Introducing latent variables to represent topics, known as topic models. [Blei et al., 2003]
  - Replacing $p(w_i|w_1, w_2, \ldots, w_{i-2}, w_{i-1})$ with a predictive neural network and an "embedding layer" to better represent larger contexts and leverage similarities between words in the context. [Bengio et al., 2000]

- Modern language models utilize deep neural networks in their backbone and have a vast parameter space.

Capítulo 4

Text Classification and Naïve Bayes

- Classification lies at the heart of both human and machine intelligence.
- Deciding what letter, word, or image has been presented to our senses, recognizing faces or voices, sorting mail, assigning grades to homeworks.
- These are all examples of assigning a category to an input.
- The goal of classification is to take a single observation, extract some useful features, and thereby classify the observation into one of a set of discrete classes.
- Most cases of classification in language processing are done via supervised machine learning.
- This slides are based on the course material by Daniel Jurafsky : `https://web.stanford.edu/~jurafsky/slp3/4.pdf`

Example 1: Spam Classification

Subject: **Important notice!**
From: Stanford University <newsforum@stanford.edu>
Date: October 28, 2011 12:34:16 PM PDT
To: undisclosed-recipients:;

Greats News!

You can now access the latest news by using the link below to login to Stanford University News Forum.

http://www.123contactform.com/contact-form-StanfordNew1-236335.html

Click on the above link to login for more information about this new exciting forum. You can also copy the above link to your browser bar and login for more information
about the new services.

Example 2: Who wrote which Federalist papers?

- 1787-8: Anonymous essays attempted to convince New York to ratify the U.S Constitution: Jay, Madison, Hamilton.
- Authorship of 12 of the letters is in dispute.

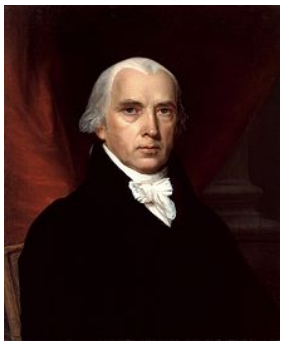


Figura 4.1: James Madison

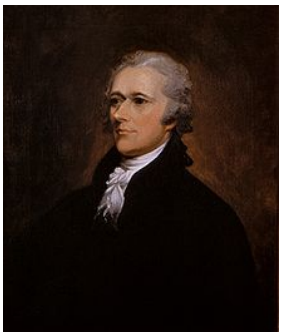
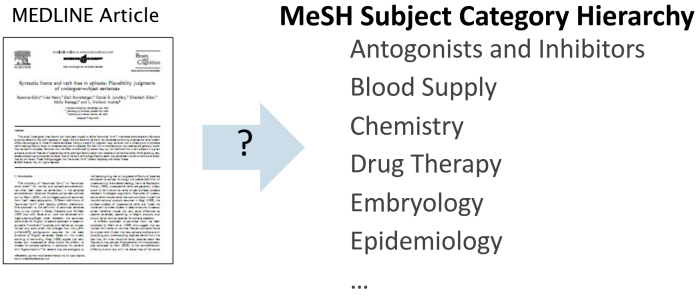


Figura 4.2: Alexander Hamilton

- 1963: Solved by Mosteller and Wallace using Bayesian methods.

Example 3: What is the subject of this medical article?



Example 4: Positive or negative movie review?

- + ...zany characters and richly applied satire, and some great plot twists
- - It was pathetic. The worst part about it was the boxing scenes...
- + ...awesome caramel sauce and sweet toasty almonds. I love this place!
- - ...awful pizza and ridiculously overpriced...

Why sentiment analysis?

- Movie: Is this review positive or negative?
- Products: What do people think about the new iPhone?
- Public sentiment: How is consumer confidence?
- Politics: What do people think about this candidate or issue?
- Prediction: Predict election outcomes or market trends from sentiment.

Basic Sentiment Classification Sentiment analysis is the detection of attitudes.

- Simple task we focus on in this class
    - Is the attitude of this text positive or negative?

Summary: Text Classification Text classification can be applied to various tasks, including:

- Sentiment analysis
- Spam detection
- Authorship identification
- Language identification
- Assigning subject categories, topics, or genres
- ...

4.1. Text Classification: Definition

Input:

- A document d
- A fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$

Output: A predicted class $c \in C$

4.1.1. Classification Methods: Hand-coded rules

Rules based on combinations of words or other features

- Spam: *black-list-address* OR (*"dollars" AND "you have been selected"*)
- Accuracy can be high if rules carefully refined by experts
- But building and maintaining these rules is expensive

4.1.2. Classification Methods: Supervised Machine Learning

Input:

- A document d
- A fixed set of classes $C = \{c_1, c_2, \ldots, c_J\}$
- A training set of m hand-labeled documents: $(d_1, c_1), (d_2, c_2), \ldots, (d_m, c_m)$

Output:

- A learned classifier $\gamma : d \to c$

Any kind of classifier can be used:

- Naïve Bayes
- Logistic regression
- Neural networks
- k-Nearest Neighbors

4.1.3. Supervised Learning Problems

- We have training examples $x^{(i)}$, $y^{(i)}$ for $i = 1, \ldots, m$. Each $x^{(i)}$ is an input, each $y^{(i)}$ is a label.
- Task is to learn a function f mapping inputs x to labels $f(x)$.
- Conditional models:
  - Learn a distribution $p(y|x)$ from training examples.
  - For any test input x, define $f(x) = \arg\max_y p(y|x)$.

4.1.4. Generative Models

- Given training examples $x^{(i)}$, $y^{(i)}$ for $i = 1, \ldots, m$. The task is to learn a function f that maps inputs x to labels $f(x)$.
- Generative models:
  - Learn the joint distribution $p(x, y)$ from the training examples.
  - Often, we have $p(x, y) = p(y)p(x|y)$.
  - Note: We then have

$$p(y|x) = \frac{p(y)p(x|y)}{p(x)} \quad \text{where} \quad p(x) = \sum_y p(y)p(x|y).$$

4.1.5. Classification with Generative Models

- Given training examples $x^{(i)}$, $y^{(i)}$ for $i = 1, \ldots, m$. The task is to learn a function f that maps inputs x to labels $f(x)$.
- Generative models:
  - Learn the joint distribution $p(x, y)$ from the training examples.
  - Often, we have $p(x, y) = p(y)p(x|y)$.
- Output from the model:

$$\begin{aligned} f(x) = \arg\max_y p(y|x) &= \arg\max_y \frac{p(y)p(x|y)}{p(x)} \\ &= \arg\max_y p(y)p(x|y) \end{aligned}$$

4.2. Naive Bayes Intuition

Naive Bayes is a simple ("naive") classification method based on Bayes' rule.

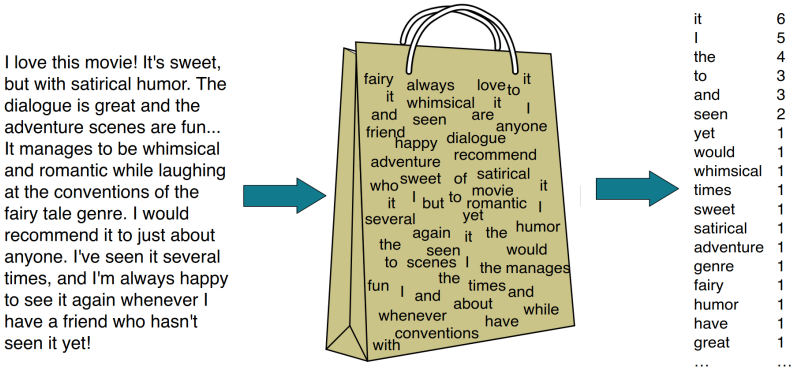
- Relies on a very simple representation of a document: *Bag of words*

The Bag of Words Representation

4.2.1. Bayes' Rule Applied to Documents and Classes

For a document d and a class c:

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)}$$



4.3. Naive Bayes Classifier

- MAP stands for "maximum a posteriori,"which represents the most likely class:

$$c_{\text{MAP}} = \arg\max_{c \in C} P(c|d)$$

- To calculate the most likely class, we apply Bayes' rule:

$$= \arg\max_{c \in C} \frac{P(d|c)P(c)}{P(d)}$$

- Finally, we can drop the denominator since it remains constant for all classes:

$$= \arg\max_{c \in C} P(d|c)P(c)$$

- To classify document d, we use the MAP estimate:

$$c_{\text{MAP}} = \arg\max_{c \in C} P(d|c)P(c)$$

- The document d is represented as a set of features $x_1, x_2, \ldots, x_n$.
- The classifier calculates the conditional probability of the features given a class and the prior probability of the class:

$$= \arg\max_{c \in C} P(x_1, x_2, \ldots, x_n|c)P(c)$$

- The term $P(x_1, x_2, \ldots, x_n|c)$ represents the "likelihood.ºf the features given the class.
- The term $P(c)$ represents the "prior"probability of the class.
- The Naïve Bayes classifier **[?]** calculates the MAP estimate by considering the likelihood and prior probabilities:

$$c_{\text{MAP}} = \arg\max_{c \in C} P(x_1, x_2, \ldots, x_n|c)P(c)$$

- The probability of the features given the class, $P(x_1, x_2, \ldots, x_n|c)$, can be estimated by counting the relative frequencies in a corpus.
- The prior probability of the class, $P(c)$, represents how often this class occurs.
- Without some simplifying assumptions, estimating the probability of every possible combination of features in $P(x_1, x_2, \ldots, x_n|c)$ would require huge numbers of parameters and impossibly large training sets.
- Naive Bayes classifiers therefore make two simplifying assumptions.

4.3.1. Multinomial Naive Bayes Independence Assumptions

- Bag of Words assumption: We assume that the position of words in the document does not matter.
- Conditional Independence assumption: We assume that the feature probabilities $P(x_i|c_j)$ are independent given the class c_j.
- In the Multinomial Naive Bayes classifier, the probability of a document with features $x_1, x_2, \ldots, x_n$ given class c can be calculated as:

$$P(x_1, x_2, \ldots, x_n|c) = P(x_1|c) \cdot P(x_2|c) \cdot P(x_3|c) \cdot \ldots \cdot P(x_n|c)$$

4.3.2. Multinomial Naive Bayes Classifier

- The Maximum A Posteriori (MAP) estimate for class c in the Multinomial Naive Bayes classifier is given by:

$$c_{\text{MAP}} = \arg\max_{c \in C} P(x_1, x_2, \ldots, x_n|c)P(c)$$

- Alternatively, we can write it as:

$$c_{\text{NB}} = \arg\max_{c \in C} P(c_j) \prod_{x \in X} P(x|c)$$

- $P(c_j)$ represents the prior probability of class c_j.
- $\prod_{x \in X} P(x|c)$ represents the likelihood of the features $x_1, x_2, \ldots, x_n$ given class c.

4.3.3. Applying Multinomial Naive Bayes Classifiers to Text Classification

- The Multinomial Naive Bayes classifier for text classification can be applied as follows:

$$c_{\text{NB}} = \arg\max_{c_j \in C} P(c_j) \prod_{i \in \text{positions}} P(x_i|c_j)$$

- c_{NB} represents the predicted class for the test document.
- C is the set of all possible classes.
- $P(c_j)$ is the prior probability of class c_j.
- $\prod_{i \in \text{positions}} P(x_i|c_j)$ calculates the likelihood of each feature x_i at position i given class c_j.
- The product is taken over all word positions in the test document.

4.3.4. Problems with Multiplying Lots of Probabilities

- Multiplying lots of probabilities can result in floating-point underflow, especially when dealing with small probabilities.
- Example: $0{,}0006 \times 0{,}0007 \times 0{,}0009 \times 0{,}01 \times 0{,}5 \times 0{,}000008 \ldots$
- Idea: Use logarithms, as $\log(ab) = \log(a) + \log(b)$.
- Instead of multiplying probabilities, we can sum the logarithms of probabilities.
- The Multinomial Naive Bayes classifier can be expressed using logarithms as follows:

$$c_{\text{NB}} = \arg\max_{c_j \in C} \left(\log(P(c_j)) + \sum_{i \in \text{position}} \log(P(x_i|c_j)) \right)$$

- By taking logarithms, we avoid the issue of floating-point underflow and perform calculations in the log space.
- The classifier becomes a linear model, where the prediction is the argmax of a sum of weights (log probabilities) and the inputs (log conditional probabilities):
- Thus, Naïve Bayes is a linear classifier, operating in the log space.

4.3.5. Learning the Multinomial Naive Bayes Model

First attempt: Maximum Likelihood Estimates

- The probabilities are estimated using the observed counts in the training data.
- The prior probability of a class c_j is estimated as:

$$\hat{P}(c_j) = \frac{N_{c_j}}{N_{\text{total}}}$$

  where N_{c_j} is the number of documents in class c_j and N_{total} is the total number of documents.

- The estimate of the probability of word w_i given class c_j is calculated as:

$$\hat{P}(w_i|c_j) = \frac{\text{count}(w_i, c_j)}{\sum_{w \in V} \text{count}(w, c_j)}$$

  where $w \in V$ represents a word in the vocabulary V.

- The denominator is the sum of counts of all words in the vocabulary within class c_j.

4.3.6. Parameter Estimation

To estimate the parameters of the Multinomial Naive Bayes model, we follow these steps:

- Create a mega-document for each topic c_j by concatenating all the documents in that topic.
- We calculate the frequency of word w_i in the mega-document, which represents the fraction of times word w_i appears among all words in the documents of topic c_j.
- The estimated probability $\hat{P}(w_i|c_j)$ of word w_i given class c_j is obtained by dividing the count of occurrences of w_i in the mega-document of topic c_j by the total count of words in the mega-document:

$$\hat{P}(w_i|c_j) = \frac{\text{count}(w_i, c_j)}{\sum_{w \in V} \text{count}(w, c_j)}$$

  Here, $\text{count}(w_i, c_j)$ represents the number of times word w_i appears in the mega-document of topic c_j, and $\text{count}(w, c_j)$ is the total count of words in the mega-document.

4.3.7. Zero Probabilities and the Issue of Unseen Words

- Consider the scenario where we have not encountered the word "fantastic" in any training documents classified as positive (thumbs-up).
- Using maximum likelihood estimation, the probability $\hat{P}(\text{"fantastic"} \mid \text{positive})$ would be calculated as:

$$\hat{P}(\text{"fantastic"} \mid \text{positive}) = \frac{\text{count}(\text{"fantastic"}, \text{positive})}{\sum_{w \in V} \text{count}(w, \text{positive})}$$

- In this case, the count of the word "fantastic" in positive documents is zero, leading to a zero probability:

$$\hat{P}(\text{"fantastic"} \mid \text{positive}) = \frac{0}{\sum_{w \in V} \text{count}(w, \text{positive})} = 0$$

- However, zero probabilities cannot be conditioned away, regardless of the other evidence present.
- This poses a problem when calculating the maximum a posteriori (MAP) estimate, which is used for classification:

$$c_{\text{MAP}} = \arg\max_c \left(\hat{P}(c) \prod_i \hat{P}(x_i \mid c) \right)$$

- With a zero probability for a word, the entire expression becomes zero, regardless of other evidence.

4.3.8. Laplace (Add-1) Smoothing for Naïve Bayes

Handling zero probabilities with Laplace (Add-1) smoothing

- To address the problem of zero probabilities, we can employ Laplace (Add-1) smoothing technique.
- The smoothed estimate $\hat{P}(w_i \mid c)$ is calculated as:

$$\hat{P}(w_i \mid c) = \frac{\text{count}(w_i, c) + 1}{\sum_{w \in V} (\text{count}(w, c) + 1)}$$

- Here, an additional count of 1 is added to both the numerator and the denominator.
- The denominator is adjusted by adding the size of the vocabulary V to ensure proper normalization.
- By doing so, we prevent zero probabilities and allow some probability mass to be distributed to unseen words.
- This smoothing technique helps to mitigate the issue of unseen words and avoids the complete elimination of certain classes during classification.

4.3.9. Multinomial Naïve Bayes: Learning

Learning the Multinomial Naïve Bayes Model

- In order to learn the parameters of the model, we need to calculate the terms $P(c_j)$ and $P(w_k \mid c_j)$.
- For each class c_j in the set of classes C, we perform the following steps:
  - Retrieve all the documents $docs_j$ that belong to class c_j.
  - Calculate the term $P(w_k \mid c_j)$ for each word w_k in the vocabulary V:

$$P(w_k \mid c_j) = \frac{n_k + \alpha}{n + \alpha \cdot |\text{Vocabulary}|}$$

    where n_k represents the number of occurrences of word w_k in the concatenated document $Text_j$.
  - Calculate the prior probability $P(c_j)$:

$$P(c_j) = \frac{|docs_j|}{|\text{total number of documents}|}$$

- To calculate $P(w_k \mid c_j)$, we need to extract the vocabulary V from the training corpus.

4.3.10. Unknown Words

Dealing with unknown words in the test data:

- When we encounter unknown words in the test data that do not appear in the training data or vocabulary, we ignore them.
- We remove these unknown words from the test document as if they were not present at all.
- We do not assign any probability to these unknown words in the classification process.

Why don't we build an unknown word model?

- Building a separate model for unknown words is not generally helpful.
- Knowing which class has more unknown words does not provide useful information for classification.

4.3.11. Stop Words

Stop words are frequently used words like "the.ªnd .ª"that are often considered to have little or no significance in text classification. Some systems choose to ignore stop words in the classification process. Here is how it is typically done:

- Sort the vocabulary by word frequency in the training set.
- Create a stopword list by selecting the top 10 or 50 most frequent words.
- Remove all stop words from both the training and test sets, treating them as if they were never there.

However, removing stop words doesn't usually improve the performance of Naive Bayes classifiers. Therefore, in practice, most Naive Bayes algorithms use all words and do not utilize stopword lists.

4.4. Worked Sentiment Example

Training data:

Category	Text
Negative	Just plain boring, entirely predictable and lacks energy.
Negative	No surprises and very few laughs.
Positive	Very powerful.
Positive	The most fun film of the summer.

Test:

Category	Text
?	Predictable with no fun.

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable ~~with~~ no fun

1. Prior from training:

$$\hat{P}(c_j) = \frac{N_{c_j}}{N_{total}}$$

P(-) = 3/5
P(+) = 2/5

2. Drop "with"

3. Likelihoods from training:

$$p(w_i|c) = \frac{count(w_i, c) + 1}{(\sum_{w \in V} count(w, c)) + |V|}$$

$$P(\text{“predictable”}|-) = \frac{1+1}{14+20} \quad P(\text{“predictable”}|+) = \frac{0+1}{9+20}$$

$$P(\text{“no”}|-) = \frac{1+1}{14+20} \quad P(\text{“no”}|+) = \frac{0+1}{9+20}$$

$$P(\text{“fun”}|-) = \frac{0+1}{14+20} \quad P(\text{“fun”}|+) = \frac{1+1}{9+20}$$

4. Scoring the test set:

$$P(-)P(S|-) = \frac{3}{5} \times \frac{2 \times 2 \times 1}{34^3} = 6.1 \times 10^{-5}$$

$$P(+)P(S|+) = \frac{2}{5} \times \frac{1 \times 1 \times 2}{29^3} = 3.2 \times 10^{-5}$$

4.5. Naive Bayes as a Language Model

- When using individual word features and considering all words in the text, naive Bayes has an important similarity to language modeling.

- Specifically, a naive Bayes model can be viewed as a set of class-specific unigram language models, in which the model for each class instantiates a unigram language model.
- The likelihood features from the naive Bayes model assign a probability to each word $P(word|c)$, and the model also assigns a probability to each sentence:

$$P(s|c) = \prod_{i \in positions} P(w_i|c)$$

Consider a naive Bayes model with the classes positive (+) and negative (-) and the following model parameters:

w	$P(w\|+)$	$P(w\|-)$
I	0.1	0.2
love	0.1	0.001
this	0.01	0.01
fun	0.05	0.005
film	0.1	0.1
...	...	...

- Each of the two columns above instantiates a language model that can assign a probability to the sentence Ï love this fun film":

$$P(\text{"I love this fun film"}|+) = 0{,}1 \times 0{,}1 \times 0{,}01 \times 0{,}05 \times 0{,}1 = 0{,}0000005$$

$$P(\text{"I love this fun film"}|-) = 0{,}2 \times 0{,}001 \times 0{,}01 \times 0{,}005 \times 0{,}1 = 0{,}0000000010$$

- As it happens, the positive model assigns a higher probability to the sentence:

$$P(s|pos) > P(s|neg)$$

- Note that this is just the likelihood part of the naive Bayes model; once we multiply in the prior, a full naive Bayes model might well make a different classification decision.

4.6. Evaluation

- Let's consider just binary text classification tasks.
- Imagine you're the CEO of Delicious Pie Company.
- You want to know what people are saying about your pies.
- So you build a "Delicious Pie"tweet detector with the following classes:
  - Positive class: tweets about Delicious Pie Co
  - Negative class: all other tweets

4.6.1. The 2-by-2 Confusion Matrix

	System Positive	**System Negative**
Gold Positive	True Positive (TP)	False Negative (FN)
Gold Negative	False Positive (FP)	True Negative (TN)

Recall (also known as **Sensitivity** or **True Positive Rate**):

$$\text{Recall} = \frac{TP}{TP + FN}$$

Precision:

$$\text{Precision} = \frac{TP}{TP + FP}$$

Accuracy:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

4.6.2. Evaluation: Accuracy

Why don't we use accuracy as our metric?
Imagine we saw 1 million tweets:

- 100 of them talked about Delicious Pie Co.
- 999,900 talked about something else.

We could build a dumb classifier that just labels every tweet "not about pie":

- It would get 99.99 % accuracy!!! Wow!!!!
- But it would be useless! It doesn't return the comments we are looking for!

That's why we use precision and recall instead.

4.6.3. Evaluation: Precision and Recall

Precision measures the percentage of items the system detected (i.e., items the system labeled as positive) that are in fact positive (according to the human gold labels).

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}}$$

Recall measures the percentage of items that were correctly identified by the system out of all the items that should have been identified.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}$$

4.6.4. Why Precision and Recall?

Consider our dumb pie-classifier that just labels nothing as .ªbout pie."

- Accuracy = 99.99 % (it correctly labels most tweets as not about pie)
- Recall = 0 (it doesn't detect any of the 100 pie-related tweets)

Precision and recall, unlike accuracy, emphasize true positives:

- They focus on finding the things that we are supposed to be looking for.

4.6.5. A Combined Measure: F-measure

The F-measure is a single number that combines precision (P) and recall (R), defined as:

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

The F-measure, defined with the parameter β, differentially weights the importance of recall and precision.

- $\beta > 1$ favors recall
- $\beta < 1$ favors precision

When $\beta = 1$, precision and recall are equal, and we have the balanced F_1 measure:

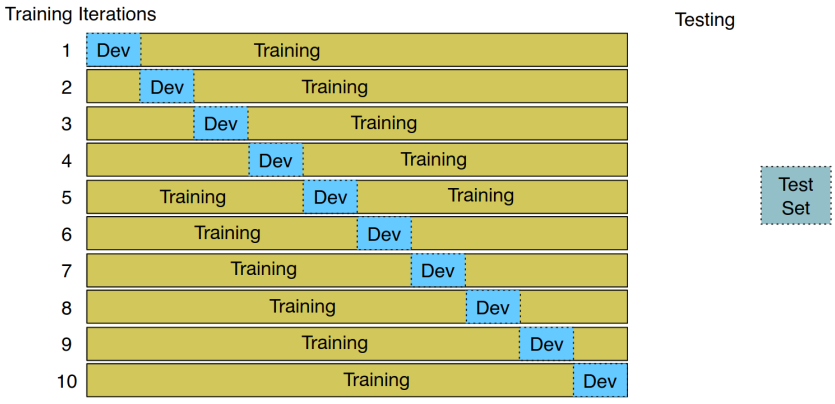
$$F_1 = \frac{2PR}{P + R}$$

4.6.6. Development Test Sets ("Devsets")

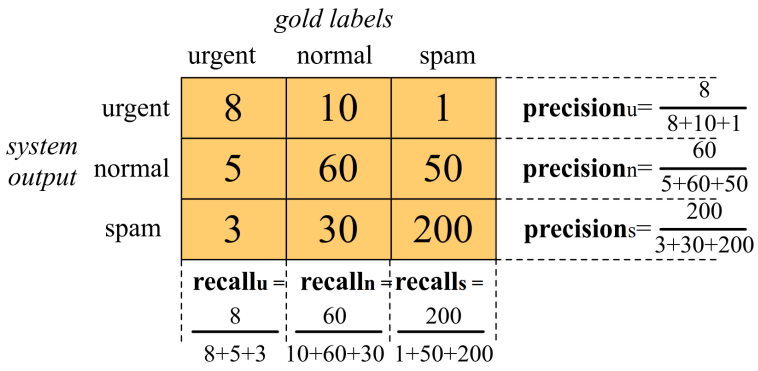
- To avoid overfitting and provide a more conservative estimate of performance, we commonly use a three-set approach: training set, devset, and testset.
  - **Training set**: Used to train the model.
  - **Devset**: Used to tune the model and select the best hyperparameters.
  - **Testset**: Used to report the final performance of the model.
- This approach ensures that the model is not tuned specifically to the test set, avoiding overfitting.
- However, it creates a paradox: we want as much data as possible for training, but also for the devset.
- How do we split the data?

4.6.7. Cross-validation: Multiple Splits

- Cross-validation allows us to use all our data for training and testing without having a fixed training set, devset, and test set.
- We choose a number k and partition our data into k disjoint subsets called folds.
- For each iteration, one fold is selected as the test set while the remaining $k - 1$ folds are used to train the classifier.
- We compute the error rate on the test set and repeat this process k times.
- Finally, we average the error rates from these k runs to obtain an average error rate.
- 10-fold cross-validation, for example, involves training 10 models on 90 % of the data and testing each model separately.
- The resulting error rates are averaged to obtain the final performance estimate.
- However, cross-validation requires the entire corpus to be blind, preventing examination of the data for feature suggestion or understanding system behavior.
- To address this, a fixed training set and test set are created, and 10-fold cross-validation is performed within the training set.
- The error rate is computed conventionally in the test set.



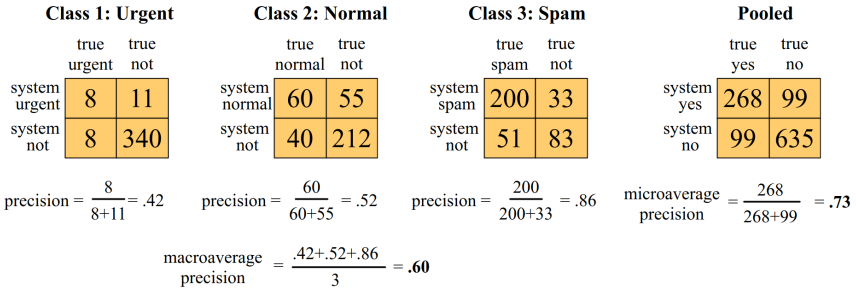
4.6.8. Confusion Matrix for 3-class classification



How to combine binary metrics (Precision, Recall, F_1) from more than 2 classes to get one metric:

- Macroaveraging:
  - Compute the performance metrics (Precision, Recall, F_1) for each class individually.
  - Average the metrics over all classes.
- Microaveraging:
  - Collect the decisions for all classes into one confusion matrix.
  - Compute Precision and Recall from the confusion matrix.

4.6.9. Macroaveraging and Microaveraging



Bibliografía

[Bender, 2013] Bender, E. M. (2013). Linguistic fundamentals for natural language processing: 100 essentials from morphology and syntax. *Synthesis lectures on human language technologies*, 6(3):1–184.

[Bengio et al., 2000] Bengio, Y., Ducharme, R., and Vincent, P. (2000). A neural probabilistic language model. *Advances in neural information processing systems*, 13.

[Blei et al., 2003] Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

[Brown et al., 1992] Brown, P. F., Desouza, P. V., Mercer, R. L., Pietra, V. J. D., and Lai, J. C. (1992). Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479.

[Chomsky, 2009] Chomsky, N. (2009). Syntactic structures. In *Syntactic Structures*. De Gruyter Mouton.

[Collobert et al., 2011] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537.

[Deng and Liu, 2018] Deng, L. and Liu, Y. (2018). *Deep Learning in Natural Language Processing*. Springer.

[Eisenstein, 2018] Eisenstein, J. (2018). Natural language processing. Technical report, Georgia Tech.

[Fromkin et al., 2018] Fromkin, V., Rodman, R., and Hyams, N. (2018). *An introduction to language*. Cengage Learning.

[Goldberg, 2016] Goldberg, Y. (2016). A primer on neural network models for natural language processing. *J. Artif. Intell. Res.(JAIR)*, 57:345–420.

[Goldberg, 2017] Goldberg, Y. (2017). Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309.

[Goodman, 2001] Goodman, J. T. (2001). A bit of progress in language modeling. *Computer Speech & Language*, 15(4):403–434.

[Johnson, 2014] Johnson, M. (2014). Introduction to computational linguistics and natural language processing (slides). 2014 Machine Learning Summer School.

[Manning et al., 2008] Manning, C. D., Raghavan, P., and Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA.

[Mohammad et al., 2013] Mohammad, S. M., Kiritchenko, S., and Zhu, X. (2013). Nrc-canada: Building the state-of-the-art in sentiment analysis of tweets. *Proceedings of the seventh international workshop on Semantic Evaluation Exercises (SemEval-2013)*.

[Nakov et al., 2013] Nakov, P., Rosenthal, S., Kozareva, Z., Stoyanov, V., Ritter, A., and Wilson, T. (2013). Semeval-2013 task 2: Sentiment analysis in twitter. In *Proceedings of the seventh international workshop on Semantic Evaluation Exercises*, pages 312–320, Atlanta, Georgia, USA. Association for Computational Linguistics.

[Read, 2005] Read, J. (2005). Using emoticons to reduce dependency in machine learning techniques for sentiment classification. In *Proceedings of the ACL Student Research Workshop*, ACLstudent '05, pages 43–48, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Salton et al., 1975] Salton, G., Wong, A., and Yang, C.-S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

[Shannon, 1951] Shannon, C. E. (1951). Prediction and entropy of printed english. *Bell system technical journal*, 30(1):50–64.

[Yule, 2016] Yule, G. (2016). *The study of language*. Cambridge university press.

[Zipf, 1935] Zipf, G. K. (1935). *The Psychobiology of Language*. Houghton-Mifflin, New York, NY, USA.