[36]

Allamanis, M., Barr, E. T., Bird, C., & Sutton, C. (2015). Suggesting accurate method and class names. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015 (pp. 38–49). New York, New York, USA: ACM Press. <https://doi.org/10.1145/2786805.2786849>

**TAGS**

OTRO MODELO AUTOMATICO; NOMBRE DE IDENTIFICADORES; ASIGNACION AUTOMATICA DE NOMBRES; NOMBRE DE METODO; NOMBRE DE CLASE; MODELO DE SUBTOKEN; MODELO DE LENGUAJE NEURONAL LOG-BILINEAR; JAVA;

**PARA ANEXAR A DOCUMENTO:**

**\*\* INICIO \*\*\***

Allamanis et al (2015) [36], propuso un modelo que genera sugerencias automáticas para nombrar un método y una clase de acuerdo a su contenido. El modelo utilizó el lenguaje neuronal log-bilineal, para capturar el contexto de la estructura del algoritmo, junto con un modelo de subtoken para realizar una división del token que analiza las relaciones entre palabras, lo que permite predecir neologismos, nombres nuevos que no habían aparecido antes. El modelo incrusta cada token (palabra) dentro de un espacio continuo de alta dimensión, como un vector. Luego, para nombrar un método (o una clase), selecciona el nombre que es más similar en este espacio de inclusión a aquellos en el cuerpo de la función. De esta manera, el modelo se da cuenta de la famosa frase de Firth: "Sabrás una palabra por la compañía que este guarda". Este eslogan encapsula la hipótesis distributiva de que palabras semánticamente similares tienden a coexistir con las mismas palabras. Las incorporaciones continuas de identificadores tienen muchas otras aplicaciones potenciales en la ingeniería de software, tales como rechazar commits cuyos nombres violan las convenciones del proyecto; exploración de antipatrones lingüísticos, tales como un getter que comienza con set y localización de características.

El modelo sugiere nombres con precisión, tanto para el problema de denominación de variables simples, como para nombrar clases y métodos logrando puntajes F1 del 60% en nombres de métodos y 55% en nombres de clase, cuando sea necesario para predecir nombres para el 20% de las declaraciones de método y clase.

**\*\* FIN \*\*\***

**NOTAS**

Modelo para sugerir Nombres para Métodos y Clases.

Introducimos un modelo de lenguaje neural probabilístico para el código fuente que está específicamente diseñado para el problema de nombrar los métodos. Nuestro modelo aprende qué nombres son semánticamente similares asignándolos a ubicaciones, llamadas incrustaciones, en un espacio continuo de alta dimensión, de tal manera que los nombres con incrustaciones similares tienden a usarse en contextos similares. Estas incrustaciones parecen contener información semántica sobre los tokens, aunque solo se aprenden de las coincidencias estadísticas de tokens. Además, presentamos una variante de nuestro modelo que, a nuestro entender, es la primera que puede proponer neologismos, nombres que no han aparecido en el corpus de entrenamiento.

**DATASET**

<http://groups.inf.ed.ac.uk/cup/naturalize/#evaldata>

**TEMAS CLAVE**

Modelo para asignar automáticamente Nombres para métodos y clases

Modelo de subtoken

Modelo de lenguaje neuronal log-bilinear

Lenguaje Java

INTRODUCCIÓN

Los nombres pobres (mal asignados) afectan negativamente la legibilidad de un código, haciendo más difícil comprenderlo y mantenerlo. Un buen nombre también puede convertirse en uno malo con el tiempo, debido a los cambios del contexto donde se utiliza.

Høst et al. : "Los métodos son las unidades más pequeñas de comportamiento agregado en la mayoría de los lenguajes de programación convencionales y, por lo tanto, la piedra angular de la abstracción".

Los nombres de métodos semánticamente distintos son las herramientas básicas para razonar sobre el comportamiento del programa. Los programadores piensan directamente en términos de estos nombres y sus composiciones, ya que un programador los eligió para las unidades en las que el programador descompuso un problema.

En este artículo, sugerimos que las herramientas estadísticas modernas nos permiten sugerir automáticamente métodos descriptivos, idiomáticos y nombres de clases a los programadores. Abordamos el problema de denominación de métodos: el problema de inferir el nombre de un método de su cuerpo (o una clase desde sus métodos).

En trabajos anteriores, presentamos el framework NATURALIZE [2], el cual aprende las convenciones de codificación utilizadas en una base de código y aborda un problema de nombres que enfrentan los programadores, el de nombrar variables, explotando la "naturalidad" o previsibilidad del código. Pero el caso de los nombres de métodos y clases es más difícil, porque además del uso también depende de su estructura interna.

los nombres de métodos y de clase a menudo son neologismos, es decir, nombres que no se ven en el corpus de entrenamiento.

El modelo asigna a cada nombre de identificador utilizado en un proyecto un vector continuo en un espacio de alta dimensión, de tal manera que los identificadores con vectores similares o "incrustaciones" tienden a aparecer en contextos similares. Luego, para nombrar un método (o una clase), seleccionamos el nombre que es más similar en este espacio de inclusión a aquellos en el cuerpo de la función. De esta manera, nuestro modelo se da cuenta de la famosa frase de Firth: "Sabrás una palabra por la compañía que este guarda". Este eslogan encapsula la hipótesis distributiva de que palabras semánticamente similares tienden a coexistir con las mismas palabras. Dos palabras son distribucionalmente similares si tienen distribuciones similares sobre las palabras circundantes.

Además, para permitirnos sugerir neologismos, presentamos un nuevo modelo de contexto de subtoken que explota la estructura interna de los nombres de los identificadores. En este modelo, predecimos los nombres dividiéndolos en partes, que llamamos subtokens, como get, create y Height, y luego pronosticamos los nombres de un subtoken a la vez. El modelo de subtoken infiere automáticamente convenciones sobre la estructura interna de los nombres de variables, como "una interfaz comienza con una I" o "una clase abstracta comienza con Abstract”.

Contribuciones son:

* Introducimos una red neuronal logarítmica bilineal para modelar contextos de código que, a diferencia de los modelos de lenguaje estándar en NLP, integra información de tokens anteriores, posteriores y no locales.
* Somos los primeros en aplicar un modelo de contexto neuronal al problema de nombrar métodos.
* Demostramos que nuestros modelos pueden sugerir nombres con precisión: para el problema de denominación de variables simples, mejoran el estado del arte, y para nombrar clases y métodos, nuestro mejor modelo logra puntajes F1 del 60% en nombres de métodos y 55% en nombres de clase, cuando sea necesario para predecir nombres para el 20% de las declaraciones de método y clase. Además, nuestro modelo de subtoken, que puede sugerir nombres nunca vistos anteriormente, alcanza una F1 del 50% cuando se requiere sugerir nombres para el 50% de las clases.

Cualquier sistema de sugerencias tiene el potencial de sufrir lo que hemos llamado el "efecto Clippy" [2], en el que demasiadas sugerencias de baja calidad alienan al usuario. Para evitar esto, nuestro modelo de sugerencias también devuelve una puntuación numérica que refleja su grado de confianza en su sugerencia; Las herramientas prácticas solo harían una sugerencia al usuario si la confianza fuera lo suficientemente alta.

MODELO DE CONTEXTO NEURONAL DEL CÓDIGO

4 modelos de lenguaje de código:

* modelo n-gram para construir intuición. Se introduce un modelado de lenguaje neuronal probabilistico.
* modelos que diseñan el nombrado de metodos y refinan el modelo neuronal.
  + modelo de contexto logbilinear (adiciona contexto y caracteristicas).
  + modelo de contexto de subtoken: adiciona subtokens y puede ser usado para generar neologismos.

LM = Modelos de lenguaje, son distribuciones de probabilidad sobre cadenas de texto de un lenguaje. El modelo asume que se trata de predecir un token “t” dada una secuencia de otro token “c”, también llamado contexto. Para el caso del problema de nombrar un método, t puede ser el identificador del token en la declaración que nombra la función, y c la secuencia que contiene todos los identificadores en el cuerpo de la función. Como no se puede guardar todos los valores de probabilidad, se hacen asunciones (asumir) simplificadas (cada LM puede hacer diferentes asunciones).

BACKGROUND

El Modelo n-gram, que es una técnica estándar en NLP y procesamiento de palabras, asume que toda la información requerida para predecir el siguiente token está contenido dentro del token previo n-1. Necesita de una tabla de V^n números, donde V es el número de posibles lexemas, que especifican la probabilidad condicional para cada posible n-gram. Son los parámetros de el modelo que nosotros aprendemos de los datos.

Modelos Logbilinear: los LMs neuronales abordan el desafío que los modelos simples n-gram tienen de hacer predicciones similares para contextos similares. Predicen el siguiente token Ym usando una red neuronal que toma un token anterior como entrada., permitiendo flexibilidad a la red para aprender cuales tokens proveen más información sobre el siguiente token y cuáles no.

Se enfocan en un tipo de LM neuronal, llamado el LM log-bilinear.

<formula de los modelos loglinear. pag 3, columna 1>

Un modelo logbilinear aprende un mapa desde cada posible etiqueta t a un vector q sub t que pertenece a R a la D. Se interpretan estas como localizaciones de cada contexto y cada etiqueta lexema en un espacio dimensional D. Estas localizaciones se llaman incrustaciones (embeddings). El modelo predice que el token t es más propenso a aparecer en el contexto c si la incrustación q sub t de el token es similar para ese r sub c de el contexto.

<formula 2, pag 3, columna 2>

Un modelo logbilinear asume que cada token y cada contexto pueden ser mapeados en un espacio dimensional D. Hay dos tipos de vectores incrustados: los aprendidos directamente (es decir, los parámetros del modelo) y los calculados a partir de los parámetros del modelo. Para indicar esta distinción, colocamos un sombrero sobre r de c para indicar que se calcula a partir de los parámetros del modelo, mientras que escribimos q de t sin sombrero para indicar que es un vector de parámetros que se aprende directamente mediante el procedimiento de entrenamiento. Estos modelos también se pueden ver como una red neuronal de tres capas.

Se usa NCE (estimación contrastante de ruido) para medir que tan bien el modelo puede diferenciar los datos reales en un conjunto de entrenamiento de los datos de fantasía que son generados por una distribución de ruido. Se optimizaron los parámetros del modelo usando descendiente de gradiente estocástico.

MODELOS DE CONTEXTO LOG BILINEAR DEL CÓDIGO.

Una nueva red neuronal, un novel de LBL LM para el código, llamado Modelo de contexto Logbilinear, el cual hace fácil explotar la información a larga distancia, es decir, tomando en cuenta todos los identificadores que aparecen en el cuerpo del método para predecir el nombre de un método.

El modelo de contexto de un código, se define como una distribución de probabilidad que sigue la formula 1 y 2, donde r con sombrero de c, es igual a r con sombrero del contexto, y se define con la formula 4.

<formula 4, página 4, columna 1>

MODELOS DE CONTEXTO DE SUB TOKEN DEL CÓDIGO

El modelo de contexto de subtoken explota el hecho de que los nombres de los identificadores a menudo se forman concatenando palabras en una frase. Un subtoken es cada palabra pequeña en un identificador. Se dividen los nombres de los identificadores en subtokens, resultando un conjunto de ellos para usar y componer nuevos identificadores.

CARACTERÍSTICAS DEL CÓDIGO FUENTE PARA LOS MODELOS DE CONTEXTO

Complejidad ciclomática.

METODOLOGÍA

RQ1. ¿Podemos identificar y extraer características de contexto de corto y largo alcance de los identificadores para nombrar?

RQ2. ¿Los identificadores contienen subestructura explotable?

RQ3. ¿Podemos sugerir con precisión los nombres de declaración de método, mirando solo el contexto del método declarado?

RQ4. ¿Podemos hacer lo mismo para los nombres de clase?

DATOS

proyectos activos de Java en GitHub en enero 22 de 2015. Seleccionando los primeros 20 proyectos, eligiendo solo 1 proyecto por dominio, que tengan mas de 50 colaboradores y más de 500 commits. Los datos se dividieron en 70% para entrenamiento y 30% para pruebas.

REPRESENTACIÓN DEL IDENTIFICADOR

Todo de los modelos que se han descrito asignan tokens, características y subtokens a incrustaciones las cuales están localizadas en un espacio continuo dimensional D. El problema está cuando el espacio dimensional es más grande, como D=50. Para ello se usan métodos de reducción de dimensionalidad. Técnicas clásicas son análisis de componentes principales (PCA) y el escalamiento multidimensional. Se emplea un método moderno llamado t-SNE. El modelo de subtokens, está aprendiendo similitudes semánticas dadas solo relaciones estadísticas entre tokens.

EVALUACIÓN

**Nombrar variables:**

Para baja frecuencia de sugerencias (alta confianza decisiones), los modelos neuronales superan las sugerencias basadas en n-gramas. Las características dan un aumento sustancial del rendimiento en comparación con los modelos que carecen de características. El modelo de subtoken funciona peor en comparación con el modelo de nivel de token para frecuencias de sugerencia superiores al 6%.

**Precisión al nombrar un método:**

el modelo neuronal es excepcionalmente bueno para predecir nombres de declaración de métodos. Cuando incluimos la exclusión de los métodos con sobreescritura, el rendimiento de todos los modelos mejora. Los modelos neurales son sustancialmente mejores para sugerir nombres de métodos, en comparación con el modelo de lenguaje de n-gramas. Agregar características aumenta el rendimiento de los modelos, lo que indica que el modelo puede usar el contexto no local para hacer mejores predicciones.

Los modelos a nivel de token no pueden hacer sugerencias más allá de una frecuencia de sugerencias del 15%. Para todos los demás tokens, los métodos a nivel de token devuelven el token especial UNK, lo que indica que los modelos esperan un neologismo que no pueden predecir. En contraste, los modelos de subtoken mantienen una buena puntuación de F1, incluso para frecuencias de sugerencia grandes. Esto se debe al hecho de que los modelos de subtoken aprenden las convenciones de nomenclatura a nivel de subtoken, capturando patrones lingüísticos [5] tales como funciones específicas que pueden contener varios subtokens.

Concluimos que podemos sugerir nombres de métodos precisos y que nuestras sugerencias son mejores que los enfoques anteriores.

PRECISIÓN EN LA DEFINICIÓN DE CLASES

Los modelos de nivel de token no pueden hacer sugerencias, siempre sugieren el token UNK. Sin embargo, el modelo de subtoken puede hacer sugerencias incluso a altas frecuencias, manteniendo un puntaje F1 de más del 40% que supera al modelo de n-gram. Gracias a la capacidad del modelo de subtoken para sugerir neologismos, el modelo de nivel de subtoken puede sugerir nombres de definición de clase que nunca ha visto antes, con una buena puntuación F1.

TRABAJOS RELACIONADOS

**Nomenclatura en la ingeniería de software:**

Arnaoudova y col. presentó un catálogo de "antipatrones lingüísticos" en el código que llevó a los desarrolladores a entender mal el código y construyó un detector de tales antipatrones [5]. Binkley utilizó parte del etiquetado de voz para encontrar nombres de campo que violen los patrones aceptados, p. el campo create\_mp4 comienza con un verbo e implica una acción que es un patrón común para un método en lugar de un campo [11]. Nuestro trabajo es complementario, ya que hacemos sugerencias de nombres cuando se encuentran errores de nombres, se producen antipatrones o se violan las reglas de nombres. De Lucio y col. intentó nombrar automáticamente los artefactos del código fuente usando LSI y LDA y descubrió que este enfoque no funciona tan bien como los métodos más simples, como el uso de palabras de clase y nombres de métodos [18]. También se han realizado muchos estudios de nomenclatura que nos dan una idea de su importancia. Butler y col. descubrió que los nombres de identificadores "defectuosos" (aquellos que violan las convenciones de nomenclatura o no siguen las pautas de práctica de codificación) están relacionados con ciertos tipos de defectos [14]. Más tarde, también examinaron las estructuras gramaticales más frecuentes de los nombres de métodos utilizando parte del etiquetado del habla [15].

Lawrie y col. [29] y Takang et al. [50] ambos realizaron estudios empíricos y concluyeron que la calidad de los nombres de los identificadores en el código tiene un profundo efecto en la comprensión del programa. Liblit y col. exploró cómo los nombres en el código "se combinan para formar frases más grandes que transmiten un significado adicional sobre el código". [30]. Arnaoudova y col. [6] estudiaron las identificaciones de identificadores, mostrando que nombrar es una parte importante de la construcción de software. Además, en una encuesta a 94 desarrolladores, encontraron que aproximadamente el 68% de los desarrolladores piensan que recomendar identificadores sería útil. Estos estudios destacan la importancia de nuestro trabajo, al poder sugerir nombres de calidad o partes de nombres. Como se espera que los nombres de método y clase indiquen su semántica, pueden verse como un caso especial de resumen de código. Haiduc y col. demostró que el resumen de texto en NL no funciona bien para el código [23] y que dichas técnicas deben adaptarse para ser efectivas. Más tarde desarrollaron resúmenes que se utilizan para mejorar la comprensión [22]. Sridhara y col. utiliza modismos y estructura en el código de métodos para generar resúmenes abstractos de alto nivel. Si bien no sugieren nombres de métodos, discuten cómo se puede extender su enfoque para proporcionarlos [47]. Sridhara también mostró cómo generar resúmenes de código apropiados para comentarios dentro del código (por ejemplo, como encabezados de método) [46, 45]. Para más trabajo en esta área, Eddy et al. proporcionar una encuesta de métodos de resumen de código [20]. Observamos que la mayoría de los estudios y enfoques en esta área se centran en los nombres de variables, campos y métodos. Aunque algunos examinan todos los identificadores en el código, no conocemos ningún trabajo que se centre en los nombres de tipo (clase) como lo hacemos nosotros.

**Modelos de lenguaje en la ingeniería de software:**

Hindle et al. y Ngyuen et al. [25, 41] utilizó modelos de n-gram para mejorar el autocompletado de código. Allamanis y Sutton [3] presentan una aplicación de modelos de código n-gram a escala. Maddison y Tarlow [31] construyeron un modelo generativo más sofisticado de código fuente utilizando modelos log-bilineales que reflejan la estructura sintáctica del código. Aunque los principios de aprendizaje automático que utilizamos son similares, su modelo difiere significativamente del nuestro, porque su propósito es construir modelos que generen código fuente en lugar de mejorar el código existente. En otras palabras, nuestro modelo es discriminatorio más que generativo. Mou et al. [40] utiliza una red neuronal convolucional para clasificar el código de los problemas de competencia de programación. Karaivanov et al. [27] combinan LMs con análisis de programas estáticos para sugerir llamadas a métodos y vacíos de relleno. Otras aplicaciones de modelos de código fuente probabilístico son la extracción de modismos de código [4] y la migración de código [27]. Estrechamente relacionado con este trabajo está nuestro trabajo anterior donde inferimos convenciones de formato y nomenclatura [2] usando LM de n-grams para sugerir cambios naturales. Raychev et al. [43] presentan un modelo probabilístico discriminativo para predecir tipos y nombres de variables en JavaScript. Por el contrario, nuestro trabajo actual introduce un modelo log-bilineal que mejora en gran medida el n-gram LM, especialmente en el método y el nombre de clase, proponiendo neologismos teniendo en cuenta subtokens y contexto no local.

**Otras aplicaciones de modelos neuronales logbilineales.**

Se han utilizado modelos logbilineales neurales en PNL para LM [37, 39] y para describir imágenes con NL [28]. Se ha demostrado que los modelos log-bilineales en PNL producen representaciones (incrustaciones) de espacio vectorial semánticamente consistentes e interesantes. Los sistemas notables incluyen word2vec [35, 36] y GloVe [42]. En contraste con estos enfoques, utilizamos una noción rica de contexto no local incorporando características específicas del código fuente mientras producimos modelos de espacio vectorial similares para nombres, variables y tipos de métodos. Además, presentamos un nuevo modelo de sub-token. Relacionado con nuestro modelo de subtoken está el trabajo de Botha y Blunsom [12] que integran representaciones morfológicas compositivas de palabras en una LM log-bilineal, pero las características morfológicas solo se usan en el contexto de una LM.

CONCLUSION

Introdujimos el problema de nombrar métodos, que determina automáticamente un nombre que describe funcionalmente a un método o una clase. El trabajo previo sobre la asignación automática de nombres [2, 43] se centra en variables locales y se basa en un contexto relativamente local. Nombrar métodos es más difícil porque requiere integrar información no local del cuerpo del método o clase. Presentamos una primera solución utilizando un modelo de lenguaje neuronal log-bilineal, que incluye características de la función que capturan el contexto de larga distancia, y un modelo de subtoken que puede predecir neologismos, nombres que no aparecieron en el conjunto de entrenamiento. El modelo incrusta cada token dentro de un espacio continuo de alta dimensión.

Las incorporaciones continuas de identificadores tienen muchas otras aplicaciones potenciales en la ingeniería de software, tales como rechazar commits cuyos nombres violan las convenciones del proyecto; exploración de antipatrones lingüísticos, tales como un getter que comienza con set [5] y localización de características. Finalmente, surge un problema similar a la nomenclatura de métodos en la NLP, a saber, el problema de generar un titular a partir del texto de un articulo [8, 19]. Es posible que modelos similares al nuestro también arrojen luz sobre ese problema.