(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 110109835 A
(43)申请公布日 2019.08.09

(21)申请号 201910368699.X

(22)申请日 2019.05.05

(71)申请人 重庆大学
地址 400044 重庆市沙坪坝区正街174号

(72)发明人 徐玲 王备 帅鉴航 何健军
杨梦宁 张小洪 杨丹 葛永新
洪明坚 王洪星 黄晟 陈飞宇

(74)专利代理机构 重庆晟轩知识产权代理事务所(普通合伙) 50238
代理人 杨晓磊

(51)Int.Cl.
G06F 11/36(2006.01)
G06K 9/62(2006.01)
G06N 3/04(2006.01)
G06N 3/08(2006.01)

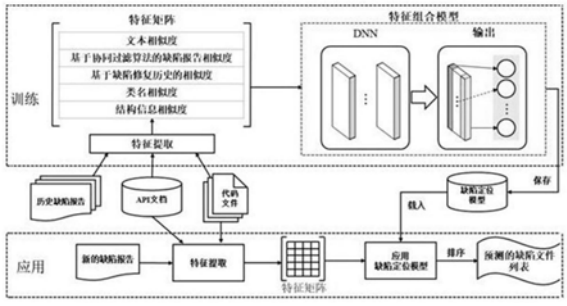
权利要求书5页 说明书18页 附图3页

(54)发明名称

一种基于深度神经网络的软件缺陷定位方法

(57)摘要

本申请公开了一种基于深度神经网络的软件缺陷定位方法(DMF-BL),该方法主要从缺陷报告和代码文件等文本数据中提取了文本相似度、结构信息相似度、基于协同过滤算法的缺陷报告相似度、基于缺陷修复历史的相似度和类名相似度五个特征,并利用深度神经网络来整合这些特征,从而捕获特征之间的非线性关系。同时,该方法在六个项目中的23000个缺陷报告上评估了软件缺陷定位的能力,结果表明,不管是Top 1、5和10中成功定位缺陷的准确率还是平均精度均值(MAP),DMF-BL的性能都要优于目前的缺陷定位技术。



CN 110109835 A

1.基于深度神经网络的软件缺陷定位方法，其特征在于，包括如下步骤：

S1:收集待测软件的相关数据

访问缺陷跟踪系统获取软件的缺陷报告，使用GIT工具获取软件的代码文件及API文档；

S2:数据预处理

对S1中收集的缺陷报告进行预处理得到如下数据：

缺陷报告集合B $=\{\overrightarrow{b_1},\overrightarrow{b_2},\overrightarrow{b_3}\ldots\overrightarrow{b_l}\ldots\}$，每个缺陷报告$\overrightarrow{b_l}=[x_1,x_2,x_3\ldots x_i\ldots]$，$x_i$代表缺陷报告中的一个单词，缺陷报告集合B的个数记为d_b；

缺陷报告修复时间向量$\overrightarrow{T}=[t_1,t_2,t_3\ldots t_l\ldots]$，$t_l$表示缺陷报告$\overrightarrow{b_l}$修复的时间；

缺陷报告提交时间向量$\overrightarrow{TS}=[ts_1,ts_2,ts_3\ldots ts_l\ldots]$，$ts_l$表示缺陷报告$b_l$提交的时间；

对S1中获得的代码文件进行预处理得到如下数据：

代码文件集合S $=\{\overrightarrow{s_1},\overrightarrow{s_2},\overrightarrow{s_3}\ldots\overrightarrow{s_m}\ldots\}$，每个代码文件$\overrightarrow{s_m}=[y_1,y_2,y_3\ldots y_j\ldots]$，$y_i$代表代码文件中的一个单词，代码文件集合S的个数记为d_s；

代码文件的名称向量$\overrightarrow{SN}=[sn_1,sn_2,sn_3\ldots sn_m\ldots]$，单词$sn_m$表示代码文件$\overrightarrow{s_m}$的名称；

对S1中获取的API文档进行预处理得到如下数据：

API文档集合D $=\{\overrightarrow{d_1},\overrightarrow{d_2},\overrightarrow{d_3}\ldots\overrightarrow{d_n}\ldots\}$，$\overrightarrow{d_i}=[z_1,z_2,z_3\ldots z_k\ldots]$，$z_k$代表API文档的一个单词；

API文档的名称向量$\overrightarrow{DA}=[da_1,da_2,da_3\ldots da_n\ldots]$，$da_n$表示API文档$\overrightarrow{d_n}$的名称；

每个缺陷报告$\overrightarrow{b_l}$对应一组标签$\overrightarrow{Tag_l}=[tag_1,tag_2,tag_3\ldots tag_i\ldots]$，$tag_i$表示缺陷报告$\overrightarrow{b_l}$所对应的代码文件的名称；

其中l、m、n，i、j和k均为正整数；

S3:为每个代码文件添加对应的API描述

遍历代码文件集合S，对于每一个$\overrightarrow{s_m}$，遍历向量$\overrightarrow{DA}$，若j＝n时，y_j＝da_n，则构成集合$S'=\{\overrightarrow{s_1'},\overrightarrow{s_2'},\overrightarrow{s_3'}\ldots\overrightarrow{s_m'}\ldots\}$，其中$\overrightarrow{s_m'}=\overrightarrow{s_m}+\overrightarrow{d_n}$；

S4:提取如下五个特征

S41:文本相似度

分别构造缺陷报告向量空间$\overrightarrow{V_B}$和和代码文件的向量空间$\overrightarrow{V_S}$，用于记录单词和单词出现的次数，$\overrightarrow{V_B}$和$\overrightarrow{V_S}$初始化为空集；

遍历缺陷报告集合B，对于每一个向量$\overrightarrow{b_l}$，都将其添加到向量空间$\overrightarrow{V_B}$中；

遍历集合S′，对于每一个向量$\overrightarrow{s_m'}$，都将其添加到向量空间$\overrightarrow{V_S}$中；

对于$\overrightarrow{V_B}$和$\overrightarrow{V_S}$，保留单词出现次数较多的单词，舍去其余记录的单词，得到新的向量空间V′$_B$和V′$_S$；

将$\overrightarrow{V_B'}$和$\overrightarrow{V_S'}$收尾相连形成新的词汇向量$\overrightarrow{V_{BS}}$，将相同的单词仅保留一个，并将单词出现的次数合并；

记d_i是缺陷报告集合B中包含单词x_i的向量的数量，单词x_i对应的逆文档频率$idf_{x_i}=\log\frac{d_b}{d_i}$，遍历缺陷报告集合B，对于每一个向量$\overrightarrow{b_l}$，设向量$\overrightarrow{v_{bl}}$为$\overrightarrow{b_l}$在向量空间$\overrightarrow{V_{BS}}$上的映射，大小为$n_{bs}$，若$x_i$属于向量空间$\overrightarrow{V_{BS}}$，单词$x_i$在向量$\overrightarrow{b_l}$中出现的次数记为$f_{x_i}$，则单词$x_i$在向量$\overrightarrow{b_l}$中出现的词频$tf(x_i,\overrightarrow{b_l})=\log(f_{x_i})+1$，单词$x_i$对应的权重大小

$$w_{x_i}=tf(x_i,\overrightarrow{b_l})\times idf_{x_i}=(\log(f_i)+1)\times\log\frac{d_b}{d_i}$$

记d_j是集合S′中包含单词y_j的向量的数量，单词y_j对应的逆文档频率$idf_{y_j}=log\frac{d_s}{d_j}$，遍历集合S′，对于每一个向量$\overrightarrow{s_m'}=[y_1,y_2,y_3\ldots y_j\ldots]$，设向量$\overrightarrow{v_{s_l'}}$为$\overrightarrow{s_l'}$在向量空间$\overrightarrow{V_{BS}}$上的映射，大小为$n_{st}$，若$y_j$属于向量空间$\overrightarrow{V_{BS}}$，单词$y_j$在向量$\overrightarrow{s_l'}$中出现的次数记为$f_j$，则单词$y_j$在向量$\overrightarrow{b_l}$中出现的词频$tf(y_j,\overrightarrow{b_l})=\log(f_j)+1$，单词$y_j$对应的权重大小

$w_{y_j}=tf(y_j,\overrightarrow{b_l})\times idf_{y_j}=(\log(f_j)+1)\times\log\frac{d_s}{d_j}$；

遍历缺陷报告集合B和集合S′，对于每个向量组$(\overrightarrow{b_l},\overrightarrow{s_m'})$，令$N_{term}$为向量$\overrightarrow{s_m'}$的长度，缺陷报告$\overrightarrow{b_l}$和代码文件$\overrightarrow{s_m'}$的文本相似度$\mathrm{sim}_t(\overrightarrow{b_l},\overrightarrow{s_m'})=\frac{1}{1+e^{-N_{term}}}\times\frac{\overrightarrow{v_{bl}}\times\overrightarrow{v_{s_l'}}}{\|\overrightarrow{v_{bl}}\|\times\|\overrightarrow{v_{s_l'}}\|}$；

S42：基于协同过滤算法的缺陷报告相似度

遍历代码文件集合S，对每一个代码文件$\overrightarrow{s_m}$，建立逆标签集合C_m，并初始化C_m为空集；

遍历缺陷报告集合B，对于每一个缺陷报告$\overrightarrow{b_l}$对应的标签$\overrightarrow{Tag_l}$，将$\overrightarrow{\mathrm{Tag_l}}$与$\overrightarrow{\mathrm{SN}}$进行对比，若$\overrightarrow{Tag_l}=\overrightarrow{SN}$，便把向量$\overrightarrow{b_l}$添加至集合$C_m$中；

遍历缺陷报告集合B和代码文件集合S，对于每个向量组$(\overrightarrow{b_l},\overrightarrow{s_m})$，遍历集合$C_m$，若$\overrightarrow{b_l}\neq\overrightarrow{b_n'}$，计算$\overrightarrow{b_l}$与$\overrightarrow{b_n'}$的余弦相似度，得相似度向量$\overrightarrow{sim_l}=[sim_1,sim_2,sim_3\ldots sim_n\ldots]$，其中$\mathrm{sim_n}$表示$\overrightarrow{b_l}$与$\overrightarrow{b_n'}$的余弦相似度；

将向量$\overrightarrow{sim_l}$按从大到小的顺序排列，得到$\overrightarrow{sim_l'}=[sim_1',sim_2',sim_3'\ldots sim_n'\ldots]$，其中每个元素已做正则化处理$sim_n'=\frac{1}{1+e^{-sim_n}}$；

计算缺陷报告$\overrightarrow{b_l}$和代码文件$\overrightarrow{s_m}$的基于协同过滤算法的缺陷报告相似度$\mathrm{sim}_{cf}(\overrightarrow{b_l},\overrightarrow{s_m})=sim_1'+\frac{1}{2}sim_2'+\frac{1}{3}sim_3'+\cdots+\frac{1}{n}\mathrm{sim_n'}$，这里$n\leqslant3$；

S43：基于缺陷修复历史的相似度

新建代码文件修复事件集合F = $\{\overrightarrow{f_1}, \overrightarrow{f_2}, \overrightarrow{f_3} \dots \overrightarrow{f_l} \dots \overrightarrow{f_{a_s}}\}$，$\overrightarrow{f_l}$表示代码文件$\overrightarrow{s_m}$被修复的历程，初始化为空向量；

遍历缺陷报告集合B，对于每一个缺陷报告$\overrightarrow{b_l}$，都有其对应的t_l和$\overrightarrow{Tag_l}$，若$\overrightarrow{Tag_l} = \overrightarrow{SN}$，则在向量$\overrightarrow{f_l}$中添加元素$t_l$；

遍历集合F，对于每一个向量$\overrightarrow{f_l}$，将元素从晚到早顺序排列，得$\overrightarrow{f_l'} = [t_1', t_2', t_3' \dots t_l' \dots]$；

遍历向量$\overrightarrow{SN}$和集合F，对于每个缺陷报告$\overrightarrow{b_l}$和代码文件$\overrightarrow{s_m}$，若$\overrightarrow{b_l}$对应的提交时间为ts_l，$\overrightarrow{s_m}$对应的修复事件向量为$\overrightarrow{f_l'} = [t_1', t_2', t_3' \dots t_l' \dots]$，将$\overrightarrow{f_l'}$中大于$ts_l$的元素删除，得$\overrightarrow{f_l''} = [t_1', t_2', t_3' \dots t_q']$，计算缺陷报告$\overrightarrow{b_l}$和代码文件$\overrightarrow{s_m}$的基于缺陷修复历史的相似度

$$\mathrm{sim}_h(\overrightarrow{b_l}, \overrightarrow{s_m}) = \sum_{p=1}^{q} \frac{1}{1 + e^{-\frac{12\times(ts_l - t_q')}{K} + (ts_l - t_1')}}$$

p为正整数，上述K值根据软件缺陷报告提交的频率决定；

S44：类名相似度

遍历缺陷报告集合B和向量$\overrightarrow{SN}$，对于每一个缺陷报告$\overrightarrow{b_l} = [x_1, x_2, x_3 \dots x_i \dots x_n]$和代码文件名称$sn_m$，若$sn_m \in \overrightarrow{b_l}$，记缺陷报告$\overrightarrow{b_l}$和代码文件$\overrightarrow{s_m}$的类名相似度$\mathrm{sim}_c(\overrightarrow{b_l}, \overrightarrow{s_m}) = |sn_m|$，反之，记$\mathrm{sim}_c(\overrightarrow{b_l}, \overrightarrow{s_m}) = 0$，将类名相似度$\mathrm{sim}_c(\overrightarrow{b_l}, \overrightarrow{s_m})$归一化；

S45：结构信息相似度

将缺陷报告集合B拆分为集合Summary和集合Description，其中Summary是缺陷报告中的summary，记$\mathrm{Summary} = [\overrightarrow{sum_1}, \overrightarrow{sum_2}, \overrightarrow{sum_3} \dots \overrightarrow{sum_l} \dots]$，Description是缺陷报告中的description，记$\mathrm{Description} = [\overrightarrow{des_1}, \overrightarrow{des_2}, \overrightarrow{des_3} \dots \overrightarrow{des_l} \dots]$，则$\overrightarrow{b_l} = \overrightarrow{sum_l} + \overrightarrow{des_l}$；

将代码文件集合S拆分成四个集合Class、Method、Var和Comment，Class是缺陷报告中的class，Method是缺陷报告中的method，Var是缺陷报告中var，Comment是缺陷报告中的comment，记$\mathrm{Class} = [\overrightarrow{cla_1}, \overrightarrow{cla_2}, \overrightarrow{cla_3} \dots \overrightarrow{cla_m} \dots]$，$\mathrm{Method} = [\overrightarrow{\mathrm{met}_1}, \overrightarrow{\mathrm{met}_2}, \overrightarrow{\mathrm{met}_3} \dots \overrightarrow{\mathrm{met}_m} \dots]$，$\mathrm{Var} = [\overrightarrow{\mathrm{var}_1}, \overrightarrow{\mathrm{var}_2}, \overrightarrow{\mathrm{var}_3} \dots \overrightarrow{\mathrm{var}_m} \dots]$，$\mathrm{Comment} = [\overrightarrow{\mathrm{com}_1}, \overrightarrow{com_2}, \overrightarrow{com_3} \dots \overrightarrow{com_m} \dots]$，则$\overrightarrow{s_m} = \overrightarrow{cla_m} + \overrightarrow{met_m} + \overrightarrow{var_m} + \overrightarrow{com_m}$；

遍历缺陷报告集合B和代码文件集合S，对于缺陷报告$\overrightarrow{b_l}$和代码文件$\overrightarrow{s_m}$，使用简单共有词方法计算如下文本相似度：

$\mathrm{sim}_1 = sim(\overrightarrow{sum_l}, \overrightarrow{cla_m})$，$\mathrm{sim}_2 = sim(\overrightarrow{sum_l}, \overrightarrow{met_m})$，$\mathrm{sim}_3 = sim(\overrightarrow{sum_l}, \overrightarrow{var_m})$，$\mathrm{sim}_4 = sim(\overrightarrow{sum_l}, \overrightarrow{com_m})$，$\mathrm{sim}_5 = sim(\overrightarrow{des_l}, \overrightarrow{cla_m})$，$\mathrm{sim}_6 = sim(\overrightarrow{des_l}, \overrightarrow{met_m})$，$\mathrm{sim}_7 = sim(\overrightarrow{des_l}, \overrightarrow{var_m})$，$\mathrm{sim}_8 = sim(\overrightarrow{des_l}, \overrightarrow{com_m})$；

计算缺陷报告$\overrightarrow{b_l}$和代码文件$\overrightarrow{s_m}$的结构信息相似度

$\mathrm{sim}_s(\overrightarrow{b_l},\overrightarrow{s_m})=\mathrm{sim}_1+\mathrm{sim}_2+\cdots+\mathrm{sim}_8$;

S5:CNN非线性组合

S51:使用步骤S4中的五个特征构造训练数据集,对于每一个缺陷报告$\overrightarrow{b_l}$,有5×d_s个特征值,构造特征值矩阵

$$\begin{bmatrix}\mathrm{sim}_t(\overrightarrow{b_l},\overrightarrow{f_1}) & \mathrm{sim}_t(\overrightarrow{b_l},\overrightarrow{f_2}) & \cdots & \mathrm{sim}_t(\overrightarrow{b_l},\overrightarrow{f_{d_s}})\\ \mathrm{sim}_{cf}(\overrightarrow{b_l},\overrightarrow{f_1}) & \mathrm{sim}_{cf}(\overrightarrow{b_l},\overrightarrow{f_2}) & \cdots & \mathrm{sim}_{cf}(\overrightarrow{b_l},\overrightarrow{f_{d_s}})\\ \mathrm{sim}_h(\overrightarrow{b_l},\overrightarrow{f_1}) & \mathrm{sim}_h(\overrightarrow{b_l},\overrightarrow{f_2}) & \cdots & \mathrm{sim}_h(\overrightarrow{b_l},\overrightarrow{f_{d_s}})\\ \mathrm{sim}_c(\overrightarrow{b_l},\overrightarrow{f_1}) & \mathrm{sim}_c(\overrightarrow{b_l},\overrightarrow{f_2}) & \cdots & \mathrm{sim}_c(\overrightarrow{b_l},\overrightarrow{f_{d_s}})\\ \mathrm{sim}_s(\overrightarrow{b_l},\overrightarrow{f_1}) & \mathrm{sim}_s(\overrightarrow{b_l},\overrightarrow{f_2}) & \cdots & \mathrm{sim}_s(\overrightarrow{b_l},\overrightarrow{f_{d_s}})\end{bmatrix}$$

缺陷报告$\overrightarrow{b_l}$对应的标签为$\overrightarrow{Tag_l}=\left[tag_1^{b_l},tag_1^{b_l},tag_1^{b_l}\ldots tag_i^{b_l}\ldots tag_{d_s}^{b_l}\right]$;

S52:构建卷积神经网络

权重初始化采用标准正态分布;

C_1:卷积层共有12个卷积核和12个偏移量,其中

5×1的卷积核3个,分别为W_1^{51}, W_1^{52}, W_1^{53},得矩阵C_1^{51}, C_1^{52}, C_1^{53},大小为1×d_s;

4×1的卷积核3个,分别为W_1^{41}, W_1^{42}, W_1^{43},得矩阵C_1^{41}, C_1^{42}, C_1^{43},大小为2×d_s;

3×1的卷积核3个,分别为W_1^{31}, W_1^{32}, W_1^{33},得矩阵C_1^{31}, C_1^{32}, C_1^{33},大小为3×d_s;

2×1的卷积核3个,分别为W_1^{21}, W_1^{22}, W_1^{23},得矩阵C_1^{21}, C_1^{22}, C_1^{23},大小为4×d_s;

C_2:池化层采用单列最大池化策略;

C_3:采用矩阵拼接的形式构建新矩阵,共得到如下4个矩阵

C_3^1是由C_2^{51}、C_2^{41}、C_2^{31}、C_2^{21}拼接而成,大小为4×d_s;

C_3^2是由C_2^{52}、C_2^{42}、C_2^{32}、C_2^{22}拼接而成,大小为4×d_s;

C_3^3是由C_2^{53}、C_2^{43}、C_2^{33}、C_2^{23}拼接而成,大小为4×d_s;

C_3^4是由C_2^{54}、C_2^{44}、C_2^{34}、C_2^{24}拼接而成,大小为4×d_s;

C_4:卷积层共有3个卷积核W_4^{21}, W_4^{22}, W_4^{23}和3个偏移量,卷积核大小为3×1,得12个矩阵,大小都为2×d_s;

C_5:卷积层共有3个卷积核W_5^{21}, W_5^{22}, W_5^{23}和3个偏移量,卷积核大小为2×1,得36个矩阵,大小都为1×d_s;

C_6:全连接层,权重矩阵为$W_6=[w_6^1,w_6^2,w_6^3\ldots w_6^{36}]$,偏移向量$B_6$,$C_6=\sum_{i=1}^{36}\left(w_6^i\times C_5^i+B_6\right)$;

C_7:分类,将C_6展开得$C_6=\left[c_1^{b_l},c_2^{b_l},c_3^{b_l}\ldots c_j^{b_l}\ldots c_{d_s}^{b_l}\right]$,得缺陷报告$\overrightarrow{b_l}$的预测值向量$Y_1$满足

$$\boldsymbol{Y_l}=\left[y_1^{b_l},y_2^{b_l},y_3^{b_l}\ldots y_j^{b_l}\ldots y_{d_s}^{b_l}\right]$$

其中，$y_j^{b_l} = f\left(c_j^{b_l}\right)$，$f(x) = \frac{1}{1+e^{-x}}$

误差函数：

$$L = \frac{1}{d_s}\sum_{d_s}^{j=1} l_j^{b_l} \log y_j^{b_l}$$

采用梯度下降法更新参数；

S6：对于新的缺陷报告$\overrightarrow{b_{new}} = \left[x_1^{b_{new}}, x_2^{b_{new}}, x_3^{b_{new}} \ldots x_i^{b_{new}} \ldots\right]$，通过S4计算5个特征值，构建特征矩阵，利用S5已训练好的CNN模型，得出对应预测值向量

$$\boldsymbol{Y_{new}} = \left[y_1^{b_{new}}, y_2^{b_{new}}, y_3^{b_{new}} \ldots y_{d_s}^{b_{new}}\right]$$

对$y_1^{b_{new}}$到$y_{d_s}^{b_{new}}$进行大小排序，得可能具有缺陷的代码文件列表，排名越靠前可能性越大。

2.根据权利要求1所述的基于深度神经网络的软件缺陷定位方法，其特征在于，所述缺陷报告、代码文件以及API文档通过文本分词、去停用词和提取词干的操作进行预处理得到缺陷报告集合B，代码文件集合S以及API文档集合D。

一种基于深度神经网络的软件缺陷定位方法

技术领域

[0001] 本发明涉及软件测试技术领域，具体来说，是一种基于深度神经网络(DNN)的多特征软件缺陷定位方法。

背景技术

[0002] 对于一个大规模的软件系统，在软件开发和维护的整个生命周期，许多项目每天都会收到大量的缺陷报告。开发人员手动完成缺陷定位是一项具有挑战性且耗时的任务。自动缺陷定位研究旨在自动定位对缺陷报告负责的潜在错误文件，以帮助开发人员专注于解决错误文件。缺陷跟踪系统(例如Bugzilla和JIRA)经常被用来记录和管理缺陷。一旦发现软件项目的异常行为，开发者或者用户可以把缺陷报告提交到缺陷跟踪系统。这些缺陷报告包含了许多字段，例如摘要和详细描述，它们描述软件的一个异常行为。这些字段对被分配去修复缺陷的开发者是非常重要的。通常，为了定位到一个缺陷报告对应的代码文件，开发者需要分析缺陷报告并查看大量代码文件，以便快速有效地修复它们。不幸的是，这些缺陷报告的数量通常对于开发者来说太大了。例如，到2016年12月，Eclipse项目报告了5100万个缺陷。对于一个给定的缺陷，手动识别潜在的缺陷文件代价太大了。因此，为了减轻软件维护团队的负担，有效的自动缺陷定位方法需求很大。

[0003] 现有技术中几种自动化的缺陷定位方法已经被提出来帮助开发人员专注于潜在的缺陷文件。现有的方法可以分为三组:动态，静态和动静混合。

[0004] 动态方法通常通过收集和分析程序数据、断点和系统的执行轨迹来定位缺陷。这种方法依赖于在某些输入条件下跟踪一组成功或失败的测试用例的执行轨迹。基于频谱的缺陷定位，和基于模型的缺陷定位是两种众所周知的动态方法。动态方法通常耗时且昂贵，它的准确率高度依赖于测试套件的质量。在实际程序中，由于大多数测试套件可能没有足够的代码覆盖来定位缺陷，动态方法有可能不可取。

[0005] 另外一方面，静态方法不需要执行跟踪，并且可以在软件开发的任何阶段被运用。它们只需要缺陷报告和代码文件就能定位缺陷。信息检索(IR)是被广泛使用的静态技术，传统的基于IR的缺陷定位通常计算缺陷报告中包含的文本描述和代码文件中的标识符名称与注释之间的相似度，然后根据它们的相似度返回一组排好序的代码文件名称。为了提高基于IR 的缺陷定位的准确性，还从缺陷报告和代码文件中提取了许多其他特征，例如结构化信息检索，缺陷报告中的元数据，动态分析，版本历史等。这些结合多个特征的方法总是比仅仅使用IR相似度的方法表现得更好。

[0006] 最近，机器学习技术被用于缺陷定位研究。这些方法通常采用训练好的机器学习模型，将缺陷报告的主题与代码文件的主题相匹配，或者将历史修复文件作为分类标签把代码文件分成许多类。王等人(Wang S,Chollak D,Movshovitz-Attias D,et al.Bugram: bug detection with n-gram language models[C].Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering.ACM,2016:708-719)使用n-gram语言模型生成可能的缺陷列表。叶等人(Ye X,Bunescu R,Liu C.Mapping bug

reports to relevant files:A ranking model,a fine-grained benchmark,and feature evaluation[J].IEEE Transactions on Software Engineering, 2016,42(4):379-402.)使用learning to rank方法对从代码文件、API描述、缺陷修复和代码变更历史中提取出的19个特征进行自适应排名。最近,深度学习被用于处理一些软件工程问题。霍等人(Huo X,Li M,Zhou Z H.Learning Unified Features from Natural and Programming Languages for Locating Buggy Source Code[C]//IJCAI.2016:1606-1612.)尝试使用一个基于 CNN的模型来学习缺陷定位的统一功能。肖等人(Xiao Y,Keung J,Mi Q,et al.Improving Bug Localization with an Enhanced Convolutional Neural Network[C]//Asia-Pacific Software Engineering Conference(APSEC),2017 24th.IEEE,2017:338-347)将增强的CNN、新的 rTF-IDuF方法和与word2vec技术结合起来,以提高缺陷定位的性能。Lam等人(Lam A N, Nguyen AT,Nguyen H A,et al.Bug localization with combination of deep learning and information retrieval[C]//Program Comprehension(ICPC),2017IEEE/ACM 25th International Conference on.IEEE,2017:218-229.)结合了DNN和基于信息检索的方法来定位有缺陷的文件。

[0007] 尽管现有技术中已经提出了许多缺陷定位的方法,有些取得了一定的效果,但是实际缺点定位非常复杂和耗时,对于实际应用来说,定位准确性依然很差,缺陷报告中的自然语言文本与代码中的编程语言之间存在着显著的固有词汇不匹配。目前的实证研究表明,缺陷定位的准确性依赖于缺陷报告和代码文件之间的多个特征提取并以适当的方式组合这些特征,这将提高缺陷定位的性能。

发明内容

[0008] 针对现有技术中存在的软件缺陷定位复杂耗时,定位准确性差的问题,本发明提供一种使用多特征组合的基于深度神经网络的软件缺陷定位方法(DMF-BL,multiple feature bug localization based on deep neural network),该方法仅提取了文本相似度、结构信息相似度、基于协同过滤算法的缺陷报告相似度、基于缺陷修复历史的相似度和类名相似度五个特征,取得了优异的软件缺陷定位性能。

[0009] 为实现上述技术目的,本发明采用的技术方案如下:

[0010] 一种基于深度神经网络的软件缺陷定位方法,包括如下步骤:

[0011] S1:收集待测软件的相关数据

[0012] 访问缺陷跟踪系统获取软件的缺陷报告,使用GIT工具获取软件的代码文件及API文档;

[0013] S2:数据预处理

[0014] 对S1中收集的缺陷报告进行预处理得到如下数据:

[0015] 缺陷报告集合B $=\{\overrightarrow{b_1},\overrightarrow{b_2},\overrightarrow{b_3}\ldots\overrightarrow{b_l}\ldots\}$,每个缺陷报告$\overrightarrow{b_l}=[x_1,x_2,x_3\ldots x_i\ldots]$,$x_i$代表缺陷报告中的一个单词,缺陷报告集合B的个数记为d_b;

[0016] 缺陷报告修复时间向量$\overrightarrow{T}=[t_1,t_2,t_3\ldots t_l\ldots]$,$t_l$表示缺陷报告$\overrightarrow{b_l}$修复的时间;

[0017] 缺陷报告提交时间向量$\overrightarrow{TS}=[ts_1,ts_2,ts_3\ldots ts_l\ldots]$,$ts_l$表示缺陷报告$b_l$提交的时间;

[0018] 对S1中获得的代码文件进行预处理得到如下数据：

[0019] 代码文件集合$S=\{\overrightarrow{s_1},\overrightarrow{s_2},\overrightarrow{s_3}\ldots\overrightarrow{s_m}\ldots\}$，每个代码文件$\overrightarrow{s_m}=[y_1,y_2,y_3\ldots y_j\ldots]$，$y_j$代表代码文件中的一个单词，代码文件集合S的个数记为d_s；

[0020] 代码文件的名称向量$\overrightarrow{SN}=[sn_1,sn_2,sn_3\ldots sn_m\ldots]$，单词$sn_m$表示代码文件$\overrightarrow{s_m}$的名称；

[0021] 对S1中获取的API文档进行预处理得到如下数据：

[0022] API文档集合$D=\{\overrightarrow{d_1},\overrightarrow{d_2},\overrightarrow{d_3}\ldots\overrightarrow{d_n}\ldots\}$，$\overrightarrow{d_l}=[z_1,z_2,z_3\ldots z_k\ldots]$，$z_k$代表API文档的一个单词；

[0023] API文档的名称向量$\overrightarrow{DA}=[da_1,da_2,da_3\ldots da_n\ldots]$，$da_n$表示API文档$\overrightarrow{d_n}$的名称；

[0024] 每个缺陷报告$\overrightarrow{b_l}$对应一组标签$\overrightarrow{Tag_l}=[tag_1,tag_2,tag_3\ldots tag_i\ldots]$，$tag_i$表示缺陷报告$\overrightarrow{b_l}$所对应的代码文件的名称；

[0025] 其中l、m、n，i、j和k均为正整数；

[0026] S3：为每个代码文件添加对应的API描述

[0027] 遍历代码文件集合S，对于每一个$\overrightarrow{s_m}$，遍历向量$\overrightarrow{DA}$，若j＝n时，$y_j=da_n$，则构成集合$S'=\{\overrightarrow{s_1'},\overrightarrow{s_2'},\overrightarrow{s_3'}\ldots\overrightarrow{s_m'}\ldots\}$，其中$\overrightarrow{s_m'}=\overrightarrow{s_m}+\overrightarrow{d_n}$；

[0028] S4：提取如下五个特征

[0029] S41：文本相似度

[0030] 分别构造缺陷报告向量空间$\overrightarrow{V_B}$和和代码文件的向量空间$\overrightarrow{V_S}$，用于记录单词和单词出现的次数，$\overrightarrow{V_B}$和$\overrightarrow{V_S}$初始化为空集；

[0031] 遍历缺陷报告集合B，对于每一个向量$\overrightarrow{b_l}$，都将其添加到向量空间$\overrightarrow{V_B}$中（重复单词会增加向量空间中对应单词出现的次数）；遍历集合S′，对于每一个向量$\overrightarrow{s_m'}$，都将其添加到向量空间$\overrightarrow{V_S}$中；

[0032] 对于$\overrightarrow{V_B}$和$\overrightarrow{V_S}$，保留单词出现次数较多的单词，舍去其余记录的单词，得到新的向量空间V'_B和V'_S；

[0033] 将$\overrightarrow{V_B'}$和$\overrightarrow{V_S'}$收尾相连形成新的词汇向量$\overrightarrow{V_{BS}}$，将相同的单词仅保留一个，并将单词出现的次数合并；

[0034] 由于$\overrightarrow{V_B'}$和$\overrightarrow{V_S'}$有可能包含同样的单词，所以V_{BS}需要去除相同的单词，并将单词计数合并，所以$\overrightarrow{V_{BS}}$的单词数量$n_{bs}\leqslant 1000$；

[0035] 记d_i是缺陷报告集合B中包含单词x_i的向量的数量，单词x_i对应的逆文档频率$idf_{x_i}=\log\frac{d_b}{d_i}$，遍历缺陷报告集合B，对于每一个向量$\overrightarrow{b_l}$，设向量$\overrightarrow{v_{bl}}$为$\overrightarrow{b_l}$在向量空间$\overrightarrow{V_{BS}}$上的映射，大小为$n_{bs}$，若$x_i$属于向量空间$\overrightarrow{V_{BS}}$，单词$x_i$在向量$\overrightarrow{b_l}$中出现的次数记为$f_{x_i}$，则单词$x_i$在

向量$\overrightarrow{b_l}$中出现的词频$tf(x_i,\overrightarrow{b_l})=\log(f_{x_i})+1$，单词$x_i$对应的权重大小

[0036] $$w_{x_i}=tf(x_i,\overrightarrow{b_l})\times idf_{x_i}=(\log(f_i)+1)\times\log\frac{d_b}{d_i}$$

[0037] 记d_j是集合S′中包含单词y_j的向量的数量，单词y_j对应的逆文档频率$idf_{y_j}=log\frac{d_s}{d_j}$，遍历集合S′，对于每一个向量$\overrightarrow{s'_m}=[y_1,y_2,y_3\dots y_j\dots]$，设向量$\overrightarrow{v_{s'_l}}$为$\overrightarrow{s'_l}$在向量空间$\overrightarrow{V_{BS}}$上的映射，大小为$n_{st}$，若$y_j$属于向量空间$\overrightarrow{V_{BS}}$，单词$y_j$在向量$\overrightarrow{s'_l}$中出现的次数记为$f_j$，则单词$y_j$在向量$\overrightarrow{b_l}$中出现的词频$tf(y_j,\overrightarrow{b_l})=\log(f_j)+1$，单词$y_j$对应的权重大小$w_{y_j}=tf(y_j,\overrightarrow{b_l})\times idf_{y_j}=(\log(f_j)+1)\times\log\frac{d_s}{d_j}$；

[0038] 遍历缺陷报告集合B和集合S′，对于每个向量组$(\overrightarrow{b_l},\overrightarrow{s'_m})$，令$N_{term}$为向量$\overrightarrow{s'_m}$的长度，缺陷报告$\overrightarrow{b_l}$和代码文件$\overrightarrow{s'_m}$的文本相似度$\mathrm{sim}_t(\overrightarrow{b_l},\overrightarrow{s'_m})=\frac{1}{1+e^{-N_{term}}}\times\frac{\overrightarrow{v_{bl}}\times\overrightarrow{v_{s'_l}}}{\|\overrightarrow{v_{bl}}\|\times\|\overrightarrow{v_{s'_l}}\|}$；

[0039] S42：基于协同过滤算法的缺陷报告相似度

[0040] 遍历代码文件集合S，对每一个代码文件$\overrightarrow{s_m}$，建立逆标签集合C_m，C_m同时对应代码文件的名称sn_m，并初始化C_m为空集；

[0041] 遍历缺陷报告集合B，对于每一个缺陷报告$\overrightarrow{b_l}$对应的标签$\overrightarrow{\mathrm{Tag_l}}$，将$\overrightarrow{\mathrm{Tag_l}}$与$\overrightarrow{\mathrm{SN}}$进行对比，若$\overrightarrow{\mathrm{Tag_l}}=\overrightarrow{\mathrm{SN}}$，便把向量$\overrightarrow{\mathrm{b_l}}$添加至集合$C_m$中，形成$\mathrm{C_m}=\{\overrightarrow{\mathrm{b'_1}},\overrightarrow{\mathrm{b'_2}},\overrightarrow{\mathrm{b'_3}}\dots\overrightarrow{\mathrm{b'_n}}\dots\}$；

[0042] 遍历缺陷报告集合B和代码文件集合S，对于每个向量组$(\overrightarrow{\mathrm{b_l}},\overrightarrow{\mathrm{s_m}})$，遍历集合$C_m$，若$\overrightarrow{b_l}\neq\overrightarrow{b'_n}$，计算$\overrightarrow{b_l}$与$\overrightarrow{b'_n}$的余弦相似度，得相似度向量$\overrightarrow{sim_l}=[sim_1,sim_2,sim_3\dots sim_n\dots]$，其中$\mathrm{sim_n}$表示$\overrightarrow{b_l}$与$\overrightarrow{b'_n}$的余弦相似度；

[0043] 将向量$\overrightarrow{sim_l}$按从大到小的顺序排列，得到$\overrightarrow{sim'_l}=[sim'_1,sim'_2,sim'_3\dots sim'_n\dots]$，其中每个元素已做正则化处理$sim'_n=\frac{1}{1+e^{-sim_n}}$；

[0044] 计算缺陷报告$\overrightarrow{b_l}$和代码文件$\overrightarrow{s_m}$的基于协同过滤算法的缺陷报告相似度$\mathrm{sim}_{cf}(\overrightarrow{b_l},\overrightarrow{s_m})=sim'_1+\frac{1}{2}sim'_2+\frac{1}{3}sim'_3+\dots+\frac{1}{n}\mathrm{sim'_n}$，这里n≤3；

[0045] S43：基于缺陷修复历史的相似度

[0046] 新建代码文件修复事件集合$\mathrm{F}=\{\overrightarrow{f_1},\overrightarrow{f_2},\overrightarrow{f_3}\dots\overrightarrow{f_l}\dots\overrightarrow{f_{d_s}}\}$，$\overrightarrow{f_l}$表示代码文件$\overrightarrow{s_m}$被修复的历程，初始化为空向量；

[0047] 遍历缺陷报告集合B，对于每一个缺陷报告$\overrightarrow{b_l}$，都有其对应的t_1和$\overrightarrow{Tag_l}$，若$\overrightarrow{Tag_l}=\overrightarrow{SN}$，则在向量$\overrightarrow{f_l}$中添加元素$t_1$；

[0048] 遍历集合F，对于每一个向量$\overrightarrow{f_l}$，将元素从晚到早顺序排列，得$\overrightarrow{f_l'}=[t_1',t_2',t_3'\ldots t_l'\ldots]$；

[0049] 遍历向量$\overrightarrow{SN}$和集合F，对于每个缺陷报告$\overrightarrow{b_l}$和代码文件$\overrightarrow{s_m}$，若$\overrightarrow{b_l}$对应的提交时间为ts_l，$\overrightarrow{s_m}$对应的修复事件向量为$\overrightarrow{f_l'}=[t_1',t_2',t_3'\ldots t_l'\ldots]$，将$\overrightarrow{f_l'}$中大于$ts_l$的元素删除，得$\overrightarrow{f_l''}=[t_1',t_2',t_3'\ldots t_q']$，计算缺陷报告$\overrightarrow{b_l}$和代码文件$\overrightarrow{s_m}$的基于缺陷修复历史的相似度

[0050] $$\mathrm{sim}_h(\overrightarrow{b_l},\overrightarrow{s_m})=\sum_q^{p=1}\frac{1}{1+e^{-\frac{12\times(ts_l-t_q')}{K}+(ts_l-t_1')}}$$

[0051] p为正整数，上述K值根据软件缺陷报告提交的频率决定；

[0052] S44：类名相似度

[0053] 遍历缺陷报告集合B和向量$\overrightarrow{SN}$，对于每一个缺陷报告$\overrightarrow{b_l}=[x_1,x_2,x_3\ldots x_i\ldots x_n]$和代码文件名称$sn_m$，若$sn_m\in\overrightarrow{b_l}$，记缺陷报告$\overrightarrow{b_l}$和代码文件$\overrightarrow{s_m}$的类名相似度$\mathrm{sim}_c(\overrightarrow{b_l},\overrightarrow{s_m})=|sn_m|$，反之，记$\mathrm{sim}_c(\overrightarrow{b_l},\overrightarrow{s_m})=0$，将类名相似度$\mathrm{sim}_c(\overrightarrow{b_l},\overrightarrow{s_m})$归一化；

[0054] S45：结构信息相似度

[0055] 将缺陷报告集合B拆分为集合Summary和集合Description，其中Summary是缺陷报告中的summary，记$\mathrm{Summary}=[\overrightarrow{sum_1},\overrightarrow{sum_2},\overrightarrow{sum_3}\ldots\overrightarrow{sum_l}\ldots]$，Description是缺陷报告中的description，记$\mathrm{Description}=[\overrightarrow{des_1},\overrightarrow{des_2},\overrightarrow{des_3}\ldots\overrightarrow{des_l}\ldots]$，则$\overrightarrow{b_l}=\overrightarrow{sum_l}+\overrightarrow{des_l}$；

[0056] 将代码文件集合S拆分成四个集合Class、Method、Var和Comment，Class是缺陷报告中的class，Method是缺陷报告中的method，Var是缺陷报告中var，Comment是缺陷报告中的comment，记

$\mathrm{Class}=[\overrightarrow{cla_1},\overrightarrow{cla_2},\overrightarrow{cla_3}\ldots\overrightarrow{cla_m}\ldots]$，$\mathrm{Method}=[\overrightarrow{\mathrm{met}_1},\overrightarrow{\mathrm{met}_2},\overrightarrow{\mathrm{met}_3}\ldots\overrightarrow{\mathrm{met}_m}\ldots]$，

$\mathrm{Var}=[\overrightarrow{\mathrm{var}_1},\overrightarrow{\mathrm{var}_2},\overrightarrow{\mathrm{var}_3}\ldots\overrightarrow{\mathrm{var}_m}\ldots]$，$\mathrm{Comment}=[\overrightarrow{\mathrm{com}_1},\overrightarrow{com_2},\overrightarrow{com_3}\ldots\overrightarrow{com_m}\ldots]$，则

$\overrightarrow{s_m}=\overrightarrow{cla_m}+\overrightarrow{met_m}+\overrightarrow{var_m}+\overrightarrow{com_m}$；

[0057] 遍历缺陷报告集合B和代码文件集合S，对于缺陷报告$\overrightarrow{b_l}$和代码文件$\overrightarrow{s_m}$，使用简单共有词方法计算如下文本相似度：

[0058]

$\mathrm{sim}_1=sim(\overrightarrow{sum_l},\overrightarrow{cla_m})$，$\mathrm{sim}_2=sim(\overrightarrow{sum_l},\overrightarrow{met_m})$，$\mathrm{sim}_3=sim(\overrightarrow{sum_l},\overrightarrow{var_m})$，$\mathrm{sim}_4=sim(\overrightarrow{sum_l},\overrightarrow{com_m})$，$\mathrm{sim}_5=sim(\overrightarrow{des_l},\overrightarrow{cla_m})$，$\mathrm{sim}_6=sim(\overrightarrow{des_l},\overrightarrow{met_m})$，$\mathrm{sim}_7=sim(\overrightarrow{des_l},\overrightarrow{var_m})$，$\mathrm{sim}_8=sim(\overrightarrow{des_l},\overrightarrow{com_m})$；

[0059] 计算缺陷报告$\overrightarrow{b_l}$和代码文件$\overrightarrow{s_m}$的结构信息相似度

$\mathrm{sim}_s(\overrightarrow{b_l},\overrightarrow{s_m})=\mathrm{sim}_1+\mathrm{sim}_2+\cdots+\mathrm{sim}_8$；

[0060] S5:CNN非线性组合

[0061] S51:使用步骤S4中的五个特征构造训练数据集,对于每一个缺陷报告$\overrightarrow{b_l}$,有5×d_s个特征值(因为代码文件$\overrightarrow{s_m}$有d_s个,每个缺陷报告$\overrightarrow{b_l}$和代码文件$\overrightarrow{s_m}$都有五个特征值),构造特征值矩阵

[0062]
$$\begin{bmatrix} \mathrm{sim}_t(\overrightarrow{b_l},\overrightarrow{f_1}) & \mathrm{sim}_t(\overrightarrow{b_l},\overrightarrow{f_2}) & \cdots & \mathrm{sim}_t(\overrightarrow{b_l},\overrightarrow{f_{d_s}}) \\ \mathrm{sim}_{cf}(\overrightarrow{b_l},\overrightarrow{f_1}) & \mathrm{sim}_{cf}(\overrightarrow{b_l},\overrightarrow{f_2}) & \cdots & \mathrm{sim}_{cf}(\overrightarrow{b_l},\overrightarrow{f_{d_s}}) \\ \mathrm{sim}_h(\overrightarrow{b_l},\overrightarrow{f_1}) & \mathrm{sim}_h(\overrightarrow{b_l},\overrightarrow{f_2}) & \cdots & \mathrm{sim}_h(\overrightarrow{b_l},\overrightarrow{f_{d_s}}) \\ \mathrm{sim}_c(\overrightarrow{b_l},\overrightarrow{f_1}) & \mathrm{sim}_c(\overrightarrow{b_l},\overrightarrow{f_2}) & \cdots & \mathrm{sim}_c(\overrightarrow{b_l},\overrightarrow{f_{d_s}}) \\ \mathrm{sim}_s(\overrightarrow{b_l},\overrightarrow{f_1}) & \mathrm{sim}_s(\overrightarrow{b_l},\overrightarrow{f_2}) & \cdots & \mathrm{sim}_s(\overrightarrow{b_l},\overrightarrow{f_{d_s}}) \end{bmatrix}$$

[0063] 缺陷报告$\overrightarrow{b_l}$对应的标签为$\overrightarrow{Tag_l}=\left[tag_1^{b_l},tag_1^{b_l},tag_1^{b_l}\ldots tag_i^{b_l}\ldots tag_{d_s}^{b_l}\right]$;

[0064] S52:构建卷积神经网络

[0065] 权重初始化采用标准正态分布;

[0066] C_1:卷积层共有12个卷积核和12个偏移量,其中

[0067] 5×1的卷积核3个,分别为W_1^{51}, W_1^{52}, W_1^{53},得矩阵C_1^{51}, C_1^{52}, C_1^{53},大小为1×d_s;

[0068] 4×1的卷积核3个,分别为W_1^{41}, W_1^{42}, W_1^{43},得矩阵C_1^{41}, C_1^{42}, C_1^{43},大小为2×d_s;

[0069] 3×1的卷积核3个,分别为W_1^{31}, W_1^{32}, W_1^{33},得矩阵C_1^{31}, C_1^{32}, C_1^{33},大小为3×d_s;

[0070] 2×1的卷积核3个,分别为W_1^{21}, W_1^{22}, W_1^{23},得矩阵C_1^{21}, C_1^{22}, C_1^{23},大小为4×d_s;

[0071] C_2:池化层采用单列最大池化策略,如矩阵C_1^{41}池化结果为C_2^{41},大小为1×d_s;

[0072] C_3:采用矩阵拼接的形式构建新矩阵,共得到如下4个矩阵

[0073] C_3^1是由C_2^{51}、C_2^{41}、C_2^{31}、C_2^{21}拼接而成,大小为4×d_s;

[0074] C_3^2是由C_2^{52}、C_2^{42}、C_2^{32}、C_2^{22}拼接而成,大小为4×d_s;

[0075] C_3^3是由C_2^{53}、C_2^{43}、C_2^{33}、C_2^{23}拼接而成,大小为4×d_s;

[0076] C_3^4是由C_2^{54}、C_2^{44}、C_2^{34}、C_2^{24}拼接而成,大小为4×d_s;

[0077] C_4:卷积层共有3个卷积核W_4^{21}, W_4^{22}, W_4^{23}和3个偏移量,卷积核大小为3×1,得12 个矩阵,大小都为2×d_s;

[0078] C_5:卷积层共有3个卷积核W_5^{21}, W_5^{22}, W_5^{23}和3个偏移量,卷积核大小为2×1,得36 个矩阵,大小都为1×d_s;

[0079] C_6:全连接层,权重矩阵为$W_6=[w_6^1,w_6^2,w_6^3\ldots w_6^{36}]$,偏移向量$B_6$,

$C_6 = \sum_{i=1}^{36}\left(w_6^i \times C_5^i + B_6\right)$;

[0080] C_7:分类,将C_6展开得$C_6 = \left[c_1^{b_l}, c_2^{b_l}, c_3^{b_l} \ldots c_j^{b_l} \ldots c_{d_s}^{b_l}\right]$,代入下述公式,得缺陷报告$\overrightarrow{b_l}$的预测值向量

[0081] $$\boldsymbol{Y_l} = \left[y_1^{b_l}, y_2^{b_l}, y_3^{b_l} \ldots y_j^{b_l} \ldots y_{d_s}^{b_l}\right]$$

[0082] 其中,$y_j^{b_l} = f\left(c_j^{b_l}\right)$, $f(x) = \frac{1}{1+e^{-x}}$

[0083] 误差函数:

[0084] $$L = \frac{1}{d_s}\sum_{d_s}^{j=1} l_j^{b_l} \log y_j^{b_l}$$

[0085] 采用梯度下降法更新参数;

[0086] S6:对于新的缺陷报告$\overrightarrow{b_{new}} = \left[x_1^{b_{new}}, x_2^{b_{new}}, x_3^{b_{new}} \ldots x_i^{b_{new}} \ldots\right]$,通过S4计算5个特征值,构建特征矩阵,利用步骤S5已训练好的CNN模型,得出对应预测值向量

[0087] $$\boldsymbol{Y_{new}} = \left[y_1^{b_{new}}, y_2^{b_{new}}, y_3^{b_{new}} \ldots y_{d_s}^{b_{new}}\right]$$

[0088] 对$y_1^{b_{new}}$到$y_{d_s}^{b_{new}}$进行大小排序,得可能具有缺陷的代码文件列表,排名越靠前可能性越大。

[0089] 进一步限定,所述缺陷报告、代码文件以及API文档通过文本分词、去停用词和提取词干的操作进行预处理得到缺陷报告集合B,代码文件集合S以及API文档集合D。

[0090] 文本分词算法是以空格、符号或段落为间隔,将文本分为单词组;去停用词采用国际通用的停用词表;提取词干采用Porter词干提取法。

[0091] 本申请提出了软件缺陷定位方法利用缺陷报告和代码文件之间的相关性来增强缺陷定位的性能。而且,本申请只提取了5个有用的特征,包括文本相似度、结构信息相似度、基于协同过滤算法的缺陷报告相似度、基于缺陷修复历史的相似度和类名相似度。实验结果表明其性能超过了现有的方法。

[0092] 首先,本申请使用修订过的VSM(rVSM)提取特征来检测缺陷报告和代码文件之间的文本相似度。此外,API规范也被用作输入以桥接缺陷报告中的自然语言和代码文件中的编程语言之间的词汇差距。其次,从前修复过的代码文件也可能对应相似的缺陷报告。叶等人提出的协同过滤的方法使用相似缺陷报告的简单总和并不完全准确,本申请提出了一种识别可疑文件的改进方法,而不是使用简单的总和。实验证明,改进后的特征能提高缺陷定位的性能。第三,本申请使用了缺陷预测技术,它旨在预测哪个代码文件将来可能会出错。第四,如果缺陷报告在摘要或详细描述中提到类名,则可以使用类名信息来识别相应的代码文件。我们还使用类名相似度特征来定位可疑的代码文件。第五,我们整合了Saha等人提出的结构信息,如代码文件的类和方法。最后,我们用DNN来结合五个特征,利用足够的训练数据,可以从非线性结合的数据中学习特征的权重。DNN与非线性函数的组合预期比基于IR的自适应学习的线性组合表现更好。

[0093] 本发明相比现有技术,具有如下有益效果:

[0094] 1、本申请提出的基于DNN的软件缺陷定位方法提取了五个特征，包括文本相似度、结构信息相似度、基于协同过滤算法的缺陷报告相似度、基于缺陷修复历史的相似度和类名相似度，而DNN方法能够整合所有分析数据，从而捕获特征之间的非线性关系。

[0095] 2、本申请对缺陷定位方法进行了大规模的评估，在超过23000个缺陷报告上运行了该定位方法，它们来自6个开源项目，包括Eclipse、JDT、Birt、SWT、Tomcat和AspectJ。实验表明，我们的方法相比于目前最先进的方法，比如MAP和MRR，都有了大幅的提高。

附图说明

[0096] 图1为本申请基于深度神经网络的软件缺陷定位方法的基本框架图；

[0097] 图2为Porter词干提取法的流程图；

[0098] 图3为深度神经网络的结构图；

[0099] 图4为五个特征分别在6个项目上的MAP值。

具体实施方式

[0100] 为了使本领域的技术人员可以更好地理解本发明，下面结合附图和实施例对本发明技术方案进一步说明。

[0101] 图1显示了本申请DMF-BL模型的整体框架。DMF-BL模型是把一组存储在缺陷追踪系统中的历史缺陷报告、一组系统的代码文件、在代码文件中使用的API描述和一个待定位的缺陷报告作为输入。

[0102] 我们将缺陷报告、代码文件和API描述预处理后提取了五个特征，并且建立了特征向量。这些特征包括：1)文本相似度；2)结构信息相似度；3)基于协同过滤算法的缺陷报告相似度；4)基于缺陷修复历史的相似度；5)类名相似度。每一个特征会对每一个段代码文件输出一个可疑度，这五个可疑度将被输入到深度神经网络模型中，以学习并找到缺陷报告所对应的代码文件位置。神经网络模型可以获取特征间的非线性关系，而且比将特征线性加权的方法更加合适。

[0103] 在提取特征之前，首先要对软件的缺陷报告、代码文件以及API描述进行预处理。

[0104] 1、数据预处理

[0105] 数据预处理(Data Preprocessing)是指在主要的处理以前对数据进行的一些处理。在线问答社区的数据是英文文本数据，对应的数据预处理是英文文本预处理。英文文本的预处理方法和中文的有部分区别。首先，英文文本挖掘预处理一般可以不做分词(特殊需求除外)，而中文预处理分词是必不可少的一步。第二点，大部分英文文本都是uft-8的编码，这样在大多数时候处理的时候不用考虑编码转换的问题，而中文文本处理必须要处理unicode的编码问题。而英文文本的预处理也有自己特殊的地方，第三点就是拼写问题，很多时候，预处理要包括拼写检查，所以需要在预处理前加以纠正。第四点就是词干提取(Stemming)和词形还原(Lemmatization)，原因主要是英文有单数，复数和各种时态，导致一个词会有不同的形式，比如“countries”和"country"，"wolf"和"wolves"，需要将这些词用同一个词来表示。

[0106] 总的来说，对于缺陷报告、代码文件和API描述中的文本数据进行预处理，需要经过这样几个步骤：文本分词、去停用词和提取词干。下面将对这些预处理方法做一些说明。

[0107] 1.1分词算法

[0108] 文本分词是数据预处理过程中不可缺少的部分，因为在建立索引和查询的过程中，是需要使用文本中的每一个单词作为一个表征文本，以基本的表征文本为单位。分词质量对于基于词频的相关性计算来说是无比重要的，而本次研究对象是英文文本，英文语言的基本单位就是单词，所以分词相对于中文来说较容易。

[0109] 大多数分词算法是以空格、符号或段落为间隔，将文本分为单词组，使用的正则表达式如下：

[0110]
```
pattern = r'''(?x)      # set flag to allow verbose regexps
   ([A-Z]\.)+           # abbreviations, e.g. U.S.A.
 | \w+(-\w+)*           # words with optional internal hyphens
 | \$?\d+(\.\d+)?%?     # currency and percentages, e.g. $12.40, 82%
 | \.\.\.               # ellipsis
 | [][.,;"'?():-_`]     # these are separate tokens
'''
re.findall(pattern,待分词文本)
```

[0111] 1.2去停用词

[0112] 在信息检索中，为节省存储空间和提高搜索效率，在处理自然语言数据(或文本)之前或之后会自动过滤掉某些字或词，这些字或词即被称为停用词(Stop Words)，去停用词是文本数据处理阶段中很重要的一个环节。

[0113] 对于一个给定的目的，任何一类的词语都可以被选作停用词。通常意义上，停用词大致分为两类。一类是人类语言中包含的功能词，这些功能词极其普遍，与其他词相比，功能词没有什么实际含义，最普遍的功能词是限定词(“the”、“a”、“an”、“that”、和“those”)，这些词帮助在文本中描述名词和表达概念，如地点或数量。介词如：“over”，“under”，“above”等表示两个词的相对位置。这些功能词的两个特征促使在搜索引擎的文本处理过程中对其特殊对待。第一，这些功能词极其普遍。记录这些词在每一个文档中的数量需要很大的磁盘空间。第二，由于它们的普遍性和功能，这些词很少单独表达文档相关程度的信息。如果在检索过程中考虑每一个词而不是短语，这些功能词基本没有什么帮助。

[0114] 另一类词包括词汇词，比如’want’等，这些词应用十分广泛，但是对这样的词搜索引擎无法保证能够给出真正相关的搜索结果，难以帮助缩小搜索范围，同时还会降低搜索的效率，所以通常会把这些词从问题中移去，从而提高搜索性能。

[0115] 去停用词的步骤包括建立停用词表和检索停用词表两步。停用词表的建立方式有两种，人工建立和基于概率统计自动建立。人工建立指的是通过人的主观判断或根据实际经验而建立的停用词表。基于概率统计的停用此表是根据词频信息通过一定算法构建停用词表。

[0116] 1.3提取词干

[0117] 词干提取(stemming)是英文文本预处理的特色。词干提取是词形规范化处理的重要技术之一，主要应用于信息检索和文本处理。在检索系统中，对文本中的词进行词干提

取，能够减少词的数量，缩减索引文件所占空间，并且使检索不受输入检索词的特定词形的限制，扩展检索结果，提高查全率。

[0118] 词干提取主要就是根据语言形态中的规律进行处理的，去除屈折或派生形态的词缀，获得词干。因此，只有掌握语言构成特点，深入解析语言形态变化，才能发现其中规律，提高词干提取的准确性。目前，在信息检索和文本处理应用当中，词干提取还是较为浅层的词形规范化技术，不考虑词性、语义等复杂问题，主要是进行词形的统一。

[0119] 词干提取目前有3大主流算法Porter Stemming、Lovins stemmer和Lancaster Stemming。本申请主要使用Porter Stemming(Porter词干提取法)，其算法主要流程如图2所示。

[0120] 2、特征提取

[0121] DMF-BL将一个缺陷报告-代码文件对(b,s)表示为一个含有k个特征的向量i。我们提取了五个特征来捕获缺陷报告和代码文件之间的隐含关系。这些功能还通过使用项目特定的 API文档来桥接缺陷报告和代码文件之间的词汇差距。表1总结了DMF-BL模型中使用的五个特征。

[0122] 1)文本相似度;2)结构信息相似度;3)基于协同过滤算法的缺陷报告相似度;4)基于缺陷修复历史的相似度;5)类名相似度

[0123] 表1 DMF-BL模型中使用的五个特征

[0124]

Short Description	Formula
文本相似度	$\mathrm{Score}_1(\mathrm{b},s)=\frac{1}{1+\mathrm{e}^{-\mathrm{N}(\#\mathrm{terms})}}\times\frac{\mathrm{b}^{\mathrm{T}}\mathrm{s}}{\lVert\vec{b}\rVert\times\lVert\vec{s}\rVert}$
基于协同过滤算法的缺陷报告相似度	$Score_2(b,s)=\sum_{k=1}^{n}\frac{1}{k}sim(b,b_i)$
基于缺陷修复历史的相似度	$\mathrm{Score}_3(\mathrm{b},\mathrm{s})=\sum_{s\in H_k}\frac{1}{1+e^{-\frac{12t_s}{k}+w}}$
类名相似度	$Score_4(b,s)=\begin{cases}\lvert s.class\rvert & if\ s.class\in b\\ 0 & 其他\end{cases}$

[0125]

结构信息相似度	$Score_5(b,s)=\sum_{b_p\in b}\sum_{s_p\in b}sim(b_p,s_p)$

[0126] 2.1文本相似度

[0127] 通常，缺陷报告是用自然语言表示的，而代码文件用编程语言表示。我们可以将缺陷报告和代码文件看作文本文档然后计算它们之间文本相似度。经典的文本相似度测量使用VSM (vector space model)和TF-IDF将缺陷报告和代码文件建模为术语频率向量，并使用余弦相似度计算每个代码文件和缺陷报告之间的相似度。多年来，已经提出了许多方法来改进VSM 模型的性能。周(J.Zhou,H.Zhang,and D.Lo,“Where should the bugs be

fixed?-more accurate information retrieval-based bug localization based on bug reports,"in Proceedings of the 34th International Conference on Software Engineering,ser.ICSE'12.IEEE Press,2012,pp.14-24.)提出了rVSM(revised Vector Space Model),它可以通过调整大文件的排名并结合更有效的术语 -频率变量来帮助定位相关的缺陷文件。在本文中,我们采用rVSM,因为它已被证明比传统的VSM具有更好的性能。

[0128] 对于缺陷报告,我们提取了摘要和详细描述来创建特征表示。对于代码文件,除了代码文件中使用的注释和标识符之外,我们还提取了字符串文字,每个代码文件后面也添加了对应的API描述。

[0129] 在缺陷报告和代码文件进行预处理之后,所有输入文本数据都被标记为单个术语或单词。假设{x_1,x_2,…x_n}和{y_1,y_2,…y_n}分别是从缺陷报告和代码文件中提取的术语,其中n 是所提取术语的总数。为了测量缺陷报告和代码文件之间的相似度,这两种类型的术语向量应该在相同的高维空间中。因此,如果从所有的缺陷报告集合中提取出了NB个单词,B=｛b｝,并且还从所有的代码文件集合中提取了NF个单词,F=｛f｝,然后我们将其结合成大小为 n=NB+NF的单词向量,并用这个组合向量来模拟每个缺陷报告b_i(i=｛1,…,|B|｝)和每个代码文件f_j(j=｛1,…,|F|｝),使它们在相同的n维向量空间中。

[0130] $$\vec{b} = [tf(x_1)idf(x_1), tf(x_2)idf(x_2), \cdots tf(x_n)idf(x_n)]$$

[0131] $$\vec{s} = [tf(y_1)idf(y_1), tf(y_2)idf(y_2), \cdots tf(y_n)idf(y_n)] \quad (1)$$

[0132] 在传统的VSM中,缺陷报告$\vec{b}$和代码文件$\vec{s}$之间的相关性得分被称为它们对应的向量之间的标准余弦相似度,其计算方法见等式(2)。基于单词频率(tf)和逆文档频率(idf)计算出单词权重w。在rVSM中,tf(t,d)的对数变量用于帮助平滑高频术语的影响并优化经典 VSM模型。在等式(3)中,f_{td}指的是文档d中的单词t出现的次数。文档向量中的每个权重w由等式(4)计算。其中d_t表示包含单词t的文档数,d表示存储库中的文档总数。

[0133] $$Similarity(b,s) = cos(b,s) = \frac{\vec{b}\cdot\vec{s}}{\|\vec{b}\|\times\|\vec{s}\|} \quad (2)$$

[0134] $$tf(t,d) = \log(f_{td}) + 1 \quad (3)$$

[0135] $$w_{t\in d} = tf(t,d) \times idf_t = (log(f_{td}) + 1)\, log\frac{d}{d_t} \quad (4)$$

[0136] 由于较大的代码文件往往具有较高的包含缺陷的可能性。因此,在缺陷定位时,rVSM 把较大的文件排得比较靠前。在等式(5)中,#terms表示文档d中的单词总数,函数g被定义为模型文档长度,其中N(#terms)是#term的规范化值。这里,我们使用Min-Max来规范化#terms的值,并把它作为指数函数e^{-x}的输入。e^{-x}是一种逻辑函数,它确保较大的文档在排名时获得较高的分数。然后按照等式(4)计算rVSM分数。它可以直接用作计算每个代码文件和缺陷报告之间相似度分数的特征。

[0137] $$Score_1(b,s) = g(\#terms) \times cos(b,s) = \frac{1}{1+e^{-N(\#terms)}} \times \frac{b^T\cdot s}{\|\vec{b}\|\times\|\vec{s}\|} \quad (5)$$

[0138] 2.2基于协同过滤算法的缺陷报告相似度

[0139] 软件存储库中有大量历史修复缺陷报告。许多类似的缺陷报告可能与一个相同的代码文件有关。如果先前修复过的报告与当前缺陷报告在文本上相似,那么这些相似缺陷报告相关的缺陷文件也可能与当前报告相关。协同过滤是一种广泛应用于零售、社交媒体

和流媒体服务的技术，其基于这样的想法：对于某些事物感兴趣的人在其他事物中也可能具有相似的品味。它在解决历史类似缺陷报告时也是适用的，因为它们始终与相同的缺陷有关。叶等人采用协同过滤的方法来提高缺陷定位的准确性。但是，他们采用了一种简单地将相似缺陷报告相加的办法。如等式(6)所示，他们计算当前缺陷报告的文本与所有历史缺陷报告br(b,s)的摘要之间的文本相似度，这些摘要与相同的已修复代码文件有关。

[0140] $Score_2 = sim(b, br(b,s))$ (6)

[0141] 本申请针对现有协同过滤算法计算缺陷报告相似度的方法进行改进，其主要步骤如下：

[0142] 步骤1：遍历代码文件集合S，对每一个代码文件$\vec{s}$，建立逆标签集合C_m，并初始化C_m为空集；

[0143] 遍历缺陷报告集合B，对于每一个缺陷报告$\overrightarrow{b_l}$对应的标签$\overrightarrow{Tag_l}$，将$\overrightarrow{\mathrm{Tag}_l}$与$\overrightarrow{\mathrm{SN}}$进行对比，若$\overrightarrow{Tag_l} = \overrightarrow{SN}$，便把向量$\overrightarrow{b_l}$添加至集合$C_m$中；

[0144] 遍历缺陷报告集合B和代码文件集合S，对于每个向量组$(\overrightarrow{b_l}, \vec{s})$，遍历集合$C_m$，若$\overrightarrow{b_l} \neq \overrightarrow{b_n'}$，计算$\overrightarrow{b_l}$与$\overrightarrow{b_n'}$的余弦相似度，得相似度向量$\overrightarrow{sim_l} = [sim_1, sim_2, sim_3 \dots sim_n \dots]$，其中$sim_n$表示$\overrightarrow{b_l}$与$\overrightarrow{b_n'}$的余弦相似度；

[0145] 将向量$\overrightarrow{sim_l}$按从大到小的顺序排列，得到$\overrightarrow{sim_l'} = [sim_1', sim_2', sim_3' \dots sim_n' \dots]$，其中每个元素已做正则化处理$sim_n' = \frac{1}{1+e^{-sim_n}}$；

[0146] 步骤2：计算新的缺陷报告$\overrightarrow{b_{\mathrm{new}}}$和某一个代码文件$\vec{s}$之间的相似度，需要找到所有代码文件$\vec{s}$有错并已被修复的缺陷报告集合{$b_1, b_2, b_2 \cdots b_n$}；

[0147] 步骤3：取$\overrightarrow{sim_l'}$中最大的前k个相似度值带入公式(7)，计算$\overrightarrow{b_{\mathrm{new}}}$和缺陷报告集合{$b_1, b_2, b_2 \cdots b_n$}中每个缺陷报告的基于协同过滤算法的缺陷报告相似度得分公式，即

[0148] $$Score_2(b,s) = \sum_{k=1}^{n} \frac{1}{k} sim(b, b_k) \quad (7)$$

[0149] 2.3基于缺陷修复历史的相似度

[0150] 版本控制系统中由许多代码文件变更历史数据。当缺陷被发现时，开发人员需要修复缺陷文件。然而，缺陷文件可能会在修复缺陷之前又产生新的缺陷。缺陷修复历史记录提供的信息可以帮助预测容易出错的代码文件。

[0151] 缺陷预测技术可以在用户或开发人员发现异常行为之前预测缺陷文件，这可以提供额外的特征来量化代码文件的缺陷倾向。缺陷修复历史信息可以用于帮助预测容易出错的文件。被预测容易出错的文件具有较高的可疑分数。

[0152] Kim等人(D.Kim,Y.Tao,S.Kim,and A.Zeller.Where should we fix this bug?A two-phase recommendation model.IEEE Trans.Softw.Eng.,39(11):1597-1610,Nov.2013.)提出了 BugCache，它使用先前缺陷的位置信息并维护一个大多数容易出错的代码文件或方法的相对较短的列表。它创建了一个“缓存”文件，这个文件预计在特定提交时容易出错。Rahman 等人(F.Rahman,D.Posnett,A.Hindle,E.Barr,and

P.Devanbu.Bugcache for inspections:Hit or miss?In Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering,ESEC/FSE'11,pages 322-331,New York,NY,USA, 2011.ACM.)发现了一种更简单的算法,它只能根据缺陷修复提交的数量来预测容易出错的文件。实验表明,该算法更加简单有效,但是运行起来几乎和BugCache是一样的。

[0153] Lewis等人(Lewis C,Ou R.Bug prediction at google[J].URL: http://google-engtools.blogspot.in/2011/12/bug-prediction-at-goodle.Html.2011.)改变了Rahman 等人的算法并提出了一个新算法,这个算法被称为时间加权风险(TMR)。TMR通过谷歌系统上的缺陷修复提交情况来预测缺陷文件,它的结果简单快捷。因此,我们决定使用和修改这种经过良好测试的TMR方法,以便从缺陷修复历史文件中找到容易出错的文件。它的定义如下:

[0154] $$\mathrm{Score}_3(b,s)=\sum_{s\in H_k}\frac{1}{1+e^{-\frac{12t_s}{k}+w}} \quad (8)$$

[0155] $w=\min_s(t_s\in H_k)$ (9)

[0156] 其中H_k是指在提交缺陷报告前k天发现的缺陷文件集b。k是由用户指定的,参数k的设置在实验部分将会说明。值ts是缺陷修复提交的输入缺陷报告之间经过的天数。在Lewis等人提出的算法中,w定义了衰退的程度。在本申请中,w定义了代码文件的重要性,它表示缺陷修复提交和当前缺陷报告之间的最短时间,如公式(9)所示。w越大,输出越小。此算法的输出是每个代码文件对应的可疑分数。

[0157] 2.4类名相似度

[0158] 在许多缺陷报告中,我们可以发现在摘要或者详细描述中都直接提到了类名,这提供了一个有用的信号:相关的类文件可能负责这个错误报告。例如,SWT中id为255600的缺陷报告标记化后的详细描述包含类名"ViewerAttributeBean,viewer,attribute and bean",但是只有最长名称"ViewerAttributeBean"是相关文件。因此,当类名更长时,它更具体。我们将缺陷报告与每个代码文件的名称进行比较,如果能在缺陷报告中获取到名称,则根据类名的长度得到一个$\mathrm{score}_4(b,s)$的值,否则为0。

[0159] s.class表示代码文件s的类名,|s.class|是名称的长度。然后,我们根据等式(10)计算类名相似度。这个特征的值的范围可能很大。使用特征缩放可以使所有的特征处于一个相同的范围,从而使它们彼此具有可比性。Min-Max标准化用于标准化$\mathrm{Score}_4(b,s)$。

[0160] $$Score_4(b,s)=\begin{cases}|s.class| & if\ s.class\in b\\ 0 & 其他\end{cases} \quad (10)$$

[0161] 2.5结构化信息

[0162] tf-idf模型为所有的单词赋予相同的权重。但是,有时缺陷报告只与类或方法名称类似,但与其他所有内容不同,由于其数值很大,导致余弦值很小,关键信息可能会被其他标识削弱。在这种情况下,我们认为基于代码结构的结构信息检索能够实现更加准确的缺陷定位。Saha等人(R.K.Saha,M.Lease,S.Khurshid,and D.E.Perry,"Improving bug localization using structured information retrieval,"in Proceeding of 28th IEEE/ACM International Conference on Automated Software Engineering,ser.ASE'13,Silicon Valley,CA,USA,November 11-15,2013, pp.345-355.)基于结构化检索提出

了BLUiR来进行缺陷定位，以提高技术的准确性。我们采用BLUiR方法将错误报告b解析为两个字段：b.summary和b.description，并将代码文件解析为四个字段：s.class，s.method，s.variable和s.comment。这些字段中的每一个都表示为遵循相似度计算过程的向量，然后对八个相似度求和。结构组件特征得分可以如下计算：

[0163] $$Score_5(b,s)=\sum_{b_p\in b}\sum_{s_p\in b}sim(b_p,s_p) \quad (11)$$

[0164] 其中b_p是缺陷报告中的特定字段，s_p是代码文件中的特定字段，而$sim(b_p,s_p)$是b_p和s_p的向量表示的余弦相似度。输出的结构是一组可疑分数，每个可疑分数对应一个文件。

[0165] 3、深度神经网络(DNN)结构

[0166] 以合适的方式组合上述有益特征可以改善缺陷定位的性能。现有方法始终将特征与预设权重线性结合。然而，线性模型难以不捕捉特征之间的非线性关系，这就可能限制定位的性能。DNN因为其具有出色的容量，在处理输入和输出之间高复杂度的非线性关系获得了成功，受到它的启发，我们使用DNN作为非线性特征的组合器来计算最终的可疑性分数。

[0167] DNN是一种前向传递的人工神经网络，在输入和输出层之间有多个隐藏层，如图3所示，其中较高层能够组合来自较低层的特征。在本申请基于DNN的软件缺陷定位中，提取出来的特征可以作为五个输入向量并被送到输入层。DNN通过隐藏层中的非线性函数转换输入特征，然后通过输出层中的线性函数对这些要素进行分类。在DNN中，隐藏层具有抽象效果，隐藏层的数量决定了网络提取特征的处理能力。在实验中，我们发现DNN中隐藏的层越多，使用的计算资源就越多。隐藏层数超过3层以后，训练时间将大大增加。因此，我们在DNN 模型中选择3个隐藏层。

[0168] 通常，当训练样本确定时，输入和输出层的节点也可以确认；因此，确定节点数也很重要。如果隐藏的节点数量太少，则不具备必要的学习能力和信息处理能力。相反，如果隐藏的节点数量太多，网络结构的复杂性将大大增加，网络可能在学习过程中陷入局部极小。而且，网络的学习速度也会变慢。根据公式和已知条件(输入层有五个节点，输出层平均有近 3000个节点)，我们将隐藏节点的数量设置为9，12和7。

[0169] 假设l-1层有m个神经元，对于l层中第j个节点的输出，我们有等式(12)个输出层。

[0170] $$a_j^l=\sigma\left(\sum_{k=1}^{m}w_{jk}^l\,a_k^{l-1}+b_j^l\right) \quad (12)$$

[0171] 其中b_j^l是l层中第j个节点的值，a_k^{l-1}是l-1层中第k个神经元到第l层中第j个神经元的权重，w_{jk}^l是输出中第j个节点的偏移量；σ是一个类似于sigmoid或者ReLU函数的非线性函数。

[0172] 4、评估实验

[0173] 为了评估DMF-BL的有效性，在这个部分，我们在六个开源的软件项目上进行了实验，并且把它和三种目前最先进的缺陷定位算法进行了比较。

[0174] 4.1数据集

[0175] 为了进行比较，我们使用叶等人提供的相同的数据集来对Learning-to-Rank进行评估。这个数据集总共包含超过22000个缺陷报告，它们来自六个已公开的开源项目，包括：Eclipse Platform UI，JDT，Birt，SWT，Tomcat和AspectJ。表2详细描述了这个数据集。这些项目都把Bugzilla作为缺陷跟踪系统，并且把GIT作为版本控制系统(早期的版本控制从

CVS/SVN转移到GIT)。所有的缺陷报告、代码文件存储库链接、错误文件和API规范都已经在发布在http://dx.doi.org/10.6084/m9.figshare.951967。根据需要,可以自行下载。

[0176] 表2基准数据集

[0177]

Project	Time Range	#bug reports	#source files	#API entries
Eclipse	10/01-01/14	6495	3454	1314
JDT	10/01-01/14	6274	8184	1329
Birt	06/05-12/13	4178	6841	957
SWT	02/02-01/14	4151	2056	161
Tomcat	07/02-01/14	1056	1552	389
AspectJ	03/02-01/04	593	4439	54

[0178] 对于每个项目中按时间顺序排列的缺陷报告,我们把它分成两个部分,其中80%作为训练集(较旧的缺陷),另外20%作为测试集(较新的缺陷)。在检查错误分类的实验结果时,我们发现在训练集(较旧的缺陷)中找不到一些新的错误文件,但是它们却对应了测试集中许多缺陷报告(较新的缺陷)。在本文中,我们提取了缺陷修复历史和协同过滤这两个特征,并且它们两个都需要一些历史缺陷修复数据。然而,在这种情况下,我们的方法并不能对这些从未被训练的错误文件产生作用。因此,我们将在测试集中与新的java文件相关的一些缺陷报告与训练集中的缺陷报告进行了交换。这种交换所带来的性能提升将会在下面进行说明。

[0179] 4.2评估指标

[0180] 为了对我们所提出的缺陷定位方法进行有效性评估,我们采用了三种主流的评估指标:Top-N排序,平均精度(MAP)和平均倒数排名(MRR)。

[0181] Top-N排序:这个指标记录了其关联的缺陷代码文件在返回结果中前N(N-=1,5,10) 排名的缺陷数。给定一个缺陷报告,如果前N个查询结果里面包含至少一个正确的建议,那么我们认为这个缺陷定位是准确的。这个指标的值越高,则缺陷定位的性能越好。

[0182] 平均精度(MAP):MAP是最常见的评估排序方法的IR指标,它计算了所有缺陷报告的平均精度。因此,MAP强调了所有的缺陷文件而不是第一个。单个查询的平均精度计算如下:

[0183] $$MAP = \sum_{k=1}^{M} \frac{p(k)\times pos(k)}{numeber\ of\ positive\ instances} \quad (13)$$

[0184] 其中k是返回的已排序文件中的排名,M是排名文件的数量(件),pos(k)表示第k个文件是否是错误文件,p(k)是给定顶部截止排名k的精度,计算如下:

[0185] $$p(k) = \frac{\#faulty\ methods\ in\ the\ top\ k}{k} \quad (14)$$

[0186] 平均倒数排名(MRR):MRR是一个统计值,用于评估生成对于一个查询的可能响应列表的过程。一个查询的倒数排名是返回的已排序文件中第一个错误文件的位置的乘法逆。

[0187] MRR是所有查询的倒数排名的平均值:

[0188] $$MRR = \frac{1}{Q}\sum_{Q=1}^{|Q|}\frac{1}{rank_i} \quad (15)$$

[0189] 4.3实验结果

[0190] 这部分展示了DMF-BL在表1所示的六个项目上执行缺陷定位的评估结果。

[0191] 我们将DMF-BL与以下三种最新的最先进的缺陷定位方法和一个基线进行了比较：

[0192] 方法1:Leaning to Rank(Ye X,Bunescu R,Liu C.Mapping bug reports to relevant files:A ranking model,a fine-grained benchmark,and feature evaluation[J].IEEE Transactions on Software Engineering,2016,42(4):379-402.)通过代码文件的功能分解将领域知识用于方法、API描述、缺陷修复历史和代码变成历史。

[0193] 方法2:Buglocator(J.Zhou,H.Zhang,and D.Lo,“Where should the bugs be fixed?-more accurate information retrieval-based bug localization based on bug reports,”in Proceedings of the 34th International Conference on Software Engineering,ser.ICSE’12.IEEE Press,2012,pp. 14-24.)是一种总所周知的缺陷定位技术,它根据文本相似度、代码文件的大小以及以前缺陷修复的信息对代码文件进行排名。

[0194] 方法3:VSM方法根据代码文件和缺陷报告的文本相似度对其进行排名。

[0195] 为了进行比较,我们实现了用于评估其他三种方法的相同数据集。表3展示了所有程序的实验结果。通过比较每一个系统的指标,我们发现DMF-BL和Learning to Rank比BugLocator和VSM结果更好。DMF-BL只用了五种特征。相比于Learning to Rank使用了19种特征,已经少了太多。然而,在结果上,DMF-BL的表现优于Learning to Rank太多。

[0196] 在AspectJ中,DMF-BL成功定位了40.2%的缺陷;在Eclipse中找到了40.2%的错误;在Tomcat中,排名前1位的错误也找到了43.4%。对于其他指标也观察到了同样的趋势。与 Learning to Rank方法相比,在前1的准确性上提高了7.5-33.5%;在前5的准确性上,提高了4-28.4%;在前10的准确性上,提高了3-35%。和BugLocator相比,无论是前1、前5还是前10,DMF-BL都表现得更好。在MAP和MRR方面,AspectJ得分为0.40和0.45,Birt 得分为0.21和0.23,优于其他三种方法。在SWT中,DMF-BL要比其他方法好很多,MAP (0.51)比Learning to Rank高27.5%。DMF-BL的平均MRR比Learning to Rank高8.6%。

[0197] DMF-BL持续较高的MAP和MRR也表明DMF-BL产生的有缺陷文件的整体排名也比Learning to Rank、BugLocator和VSM要好。

[0198] 表3 BMF-BL与其他算法之间的性能比较

[0199]

项目	定位算法	Top 1 (%)	Top 5 (%)	Top 10 (%)	MAP	MRR
AspectJ	**BMF-BL**	40.2	56.7	74.8	0.40	0.46
	Learning-to-Rank	37.4	52.3	63.7	0.38	0.44
	BugLocator	20.1	32.4	40.0	0.22	0.32
	VSM	11.6	20.9	18.5	0.12	0.16
Birt	**BMF-BL**	17.3	37.1	51.3	0.21	0.23
	Learning to Rank	12.4	28.9	38.1	0.17	0.21
	BugLocator	11.1	24.9	32.1	0.14	0.18
	VSM	4.3	9.6	11.7	0.05	0.07
Eclipse	**BMF-BL**	40.2	71.3	80.5	0.47	0.54
	Learning to Rank	39.7	65.4	74.3	0.44	0.51
	BugLocator	28.1	50.2	60.0	0.31	0.37
	VSM	18.5	32.7	42.2	0.20	0.25
JDT	**BMF-BL**	42.7	71.4	80.5	0.45	0.53
	Learning to Rank	33.4	63.5	72.9	0.40	0.47
	BugLocator	19.1	40.2	51.2	0.23	0.30
	VSM	9.7	20.1	18.7	0.12	0.15
SWT	**BMF-BL**	41.8	73.9	85.2	0.51	0.56
	Learning to Rank	31.3	62.4	75.0	0.40	0.46
	BugLocator	19.3	38.3	51.1	0.25	0.28
	VSM	4.4	11.8	19.9	0.08	0.09
Tomcat	**BMF-BL**	43.4	74.3	82.7	0.54	0.59
	Learning to Rank	41.9	71.5	80.2	0.52	0.55
	BugLocator	36.1	66.1	72.6	0.43	0.48
	VSM	20.8	48.7	59.9	0.33	0.36

[0200] DMF-BL在缺陷定位上使用了五种不同的特征。为了评估每个特征对性能的影响，我们选择在不使用DNN的情况下使用每种特征进行排名，然后计算每个特征的MAP。图4中的结果显示了这五种特征在每一个项目上的MAP值。例如，在AspectJ中使用文本相似度特征，系统可以获得0.2644的最佳MAP。当在AspectJ中使用缺陷修复历史特征时，系统获得了0.2469的第二高MAP。另一方面，在Tomcat中使用相同的缺陷修复历史特征时，系统得到了0.0373的最低MAP值。这说明每个特征在不同的项目中扮演者不同角色。根据图4，计算缺陷报告和代码文件之间的词汇相似度的文本相似度特征是AspectJ、Eclipse和Tomcat项目最重要的特征。用于测量新缺陷报告与之前修复过的缺陷报告之间的相似度的协同过滤特征，是项目Birt、JDT和SWT的最重要的特征。总而言之，最重要的特征是文本相似度和协同过滤，其他特征提供了补充信息，进一步提高了定位的性能。

[0201] 缺陷定位是一项具有挑战性且耗时的任务。因此，对于一个给定的缺陷报告，我们需要开发一种自动缺陷定位技术。在本文中，我们提出了DMF-BL，这是一种基于深度学习的模型，它结合了缺陷报告和代码文件之间的五个特征。DMF-BL通过API规范、缺陷修复历史和代码文件的结构信息来利用项目知识。实际缺陷定位任务的实验结果表明，DML-BL在缺陷定位上比最先进的IR和机器学习技术表现得更好。

[0202] 以上对本发明提供的基于深度神经网络的多特征缺陷定位方法进行了详细介绍。具体实施例的说明只是用于帮助理解本发明的方法及其核心思想。应当指出，对于本技术领域的普通技术人员来说，在不脱离本发明原理的前提下，还可以对本发明进行若干改进

和修饰,这些改进和修饰也落入本发明权利要求的保护范围内。

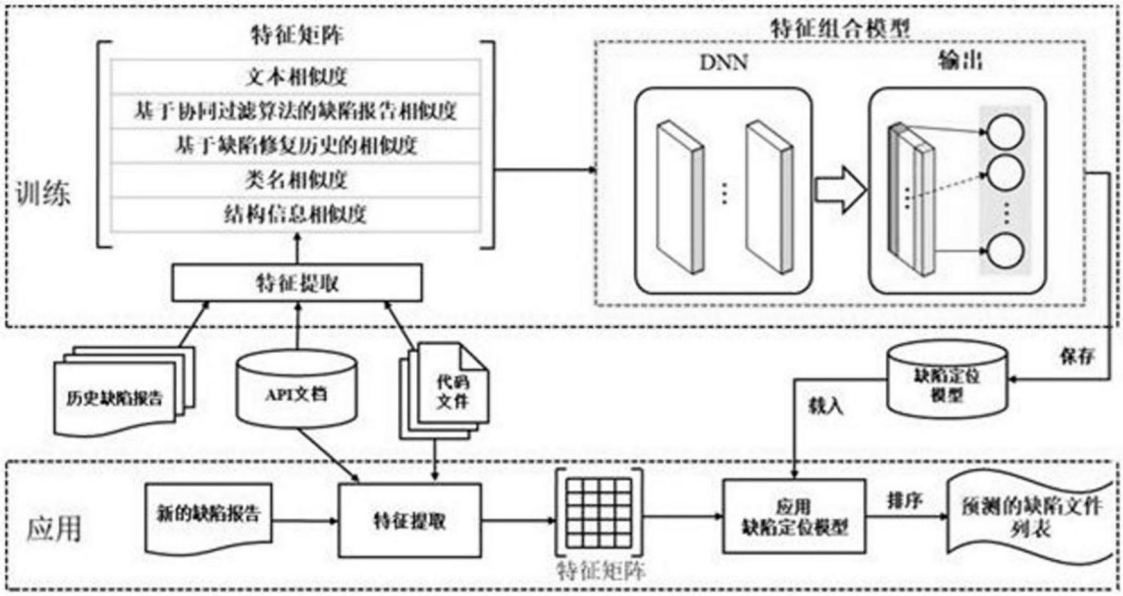

特征矩阵
文本相似度
基于协同过滤算法的缺陷报告相似度
基于缺陷修复历史的相似度
类名相似度
结构信息相似度
训练
特征提取
特征组合模型
DNN
输出
历史缺陷报告
API文档
代码文件
保存
缺陷定位模型
载入
应用
新的缺陷报告
特征提取
特征矩阵
应用缺陷定位模型
排序
预测的缺陷文件列表

图1

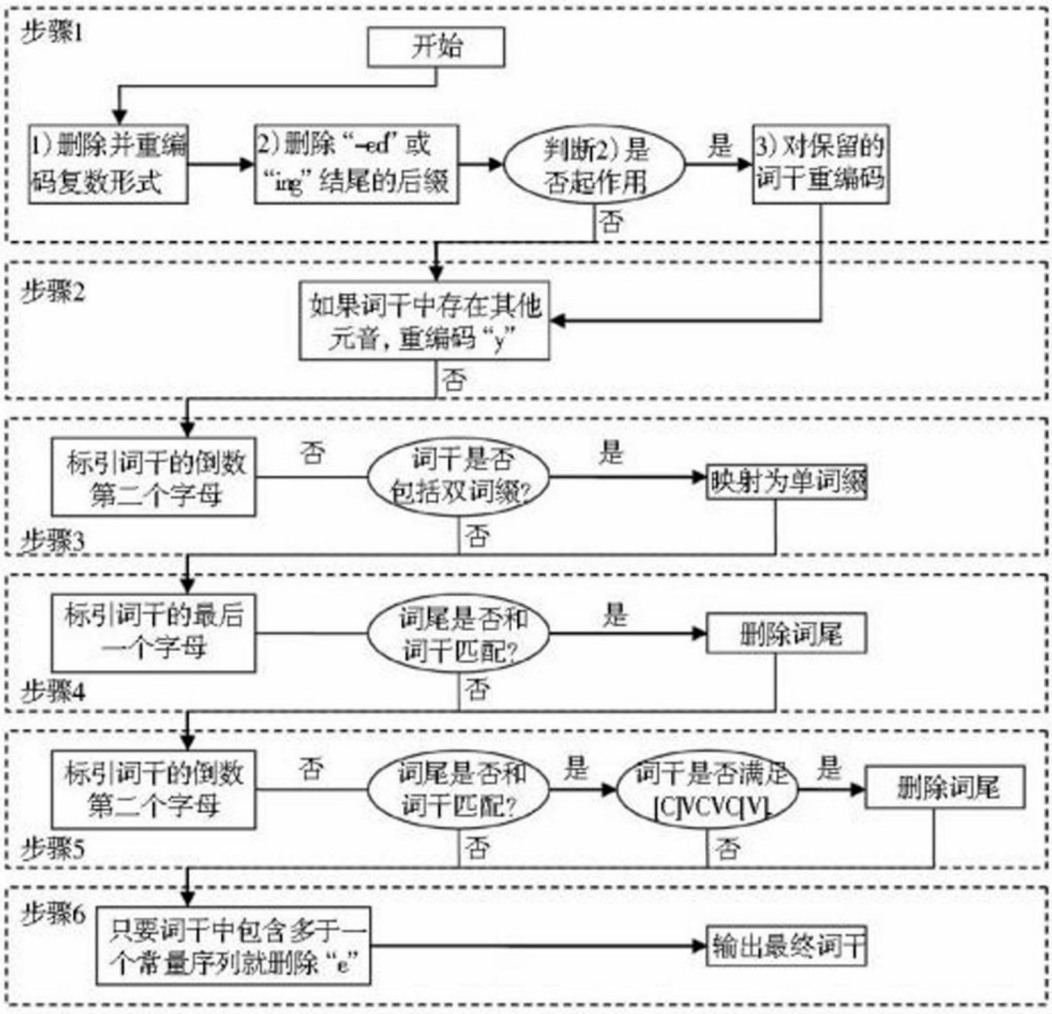

步骤1
开始
1)删除并重编码复数形式
2)删除“-ed”或“-ing”结尾的后缀
判断2)是否起作用
是
3)对保留的词干重编码
否
步骤2
如果词干中存在其他元音，重编码“y”
否
标引词干的倒数第二个字母
否
词干是否包括双词缀?
是
映射为单词缀
否
步骤3
标引词干的最后一个字母
词尾是否和词干匹配?
是
删除词尾
否
步骤4
标引词干的倒数第二个字母
否
词尾是否和词干匹配?
是
词干是否满足[C]VCVC[V]
是
删除词尾
否
否
步骤5
步骤6
只要词干中包含多于一个常量序列就删除“e”
输出最终词干

图2

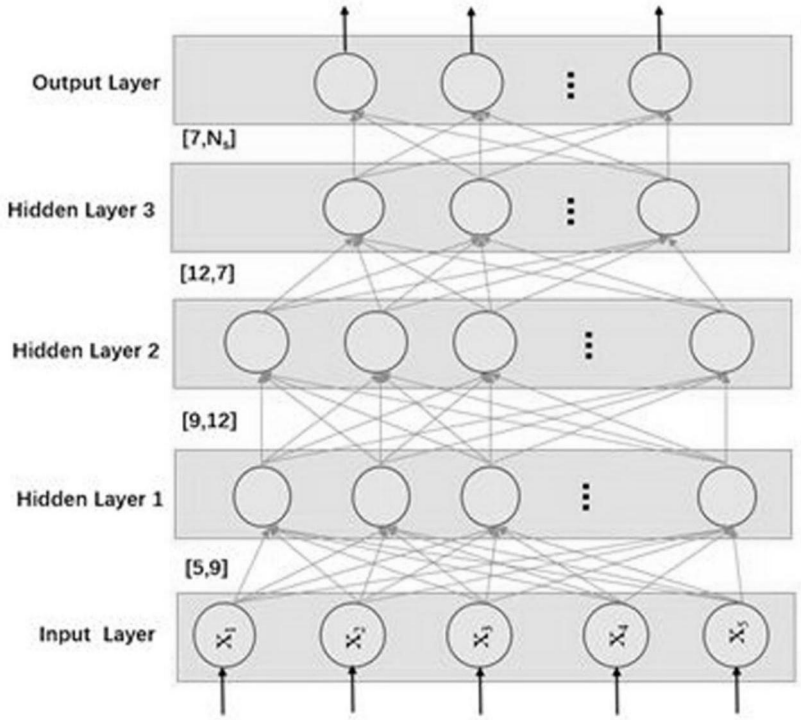

Output Layer
[7,N4]
Hidden Layer 3
[12,7]
Hidden Layer 2
[9,12]
Hidden Layer 1
[5,9]
Input Layer

图3

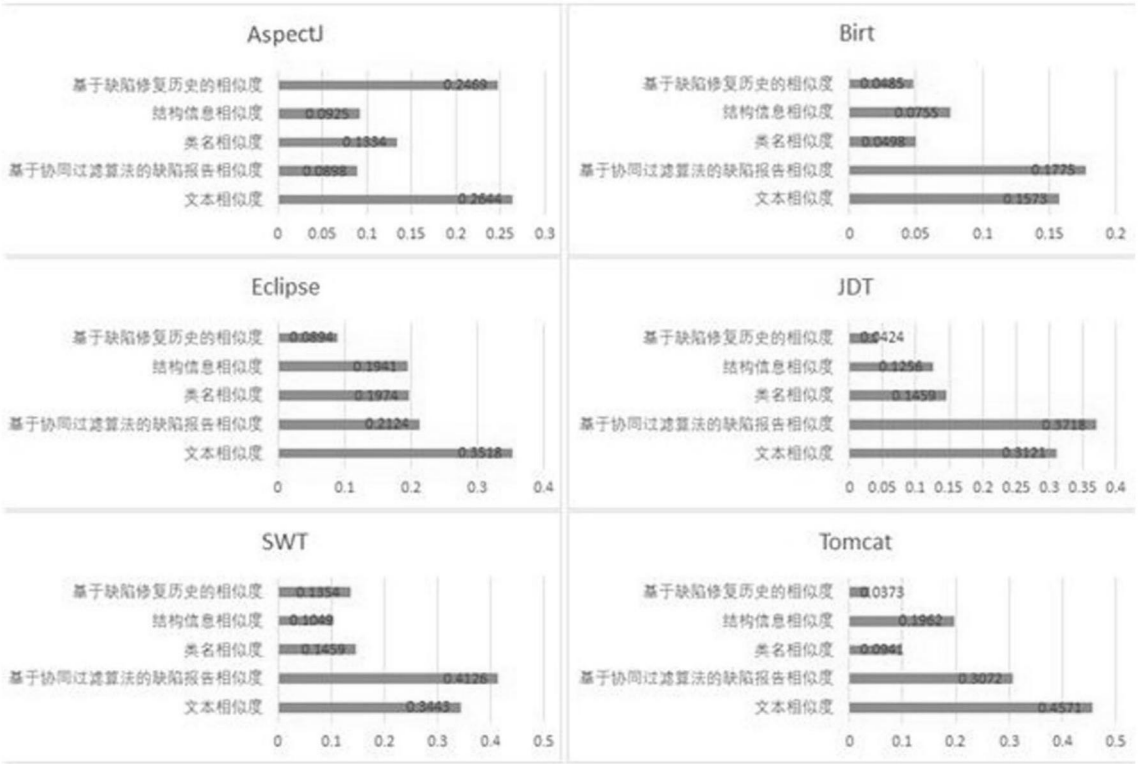

AspectJ
基于缺陷修复历史的相似度 0.2469
结构信息相似度 0.0925
类名相似度 0.1334
基于协同过滤算法的缺陷报告相似度 0.0898
文本相似度 0.2644
0 0.05 0.1 0.15 0.2 0.25 0.3
Birt
基于缺陷修复历史的相似度 0.0485
结构信息相似度 0.0755
类名相似度 0.0498
基于协同过滤算法的缺陷报告相似度 0.1775
文本相似度 0.1573
0 0.05 0.1 0.15 0.2
Eclipse
基于缺陷修复历史的相似度 0.0894
结构信息相似度 0.1941
类名相似度 0.1974
基于协同过滤算法的缺陷报告相似度 0.2124
文本相似度 0.3518
0 0.1 0.2 0.3 0.4
JDT
基于缺陷修复历史的相似度 0.0424
结构信息相似度 0.1256
类名相似度 0.1459
基于协同过滤算法的缺陷报告相似度 0.3718
文本相似度 0.3121
0 0.05 0.1 0.15 0.2 0.25 0.3 0.35 0.4
SWT
基于缺陷修复历史的相似度 0.1354
结构信息相似度 0.1049
类名相似度 0.1459
基于协同过滤算法的缺陷报告相似度 0.4126
文本相似度 0.3443
0 0.1 0.2 0.3 0.4 0.5
Tomcat
基于缺陷修复历史的相似度 0.0373
结构信息相似度 0.1962
类名相似度 0.0941
基于协同过滤算法的缺陷报告相似度 0.3072
文本相似度 0.4571
0 0.1 0.2 0.3 0.4 0.5

图4