Apprentissage Profond des Caractéristiques Structurelles et Historiques des Systèmes pour la Détection d'Anti-patrons

Antoine Barbez

Présentation de fin de maîtrise
Département de génie informatique et génie logiciel
École polytechnique de Montréal

Dirigé par:
Foutse Khomh
Yann-Gaël Guéhéneuc

13 Décembre 2018

Plan

- Définition du problème
- L’apprentissage supervisé
- Approche générale
- Première architecture: SMAD
- Deuxième architecture: CAME
- Conclusion

Anti-Patrons: analogie

Anti-Patrons: analogie

Comment fixer les roues à l'essieu?

✓ Avec des vis

✗ Avec une soudure

God Class

« *… one object with a lion's share of the responsibilities, while most other objects only hold data or execute simple processes.* »

Brown et al. (1998)

- Viole le principe de responsabilité unique

Feature Envy

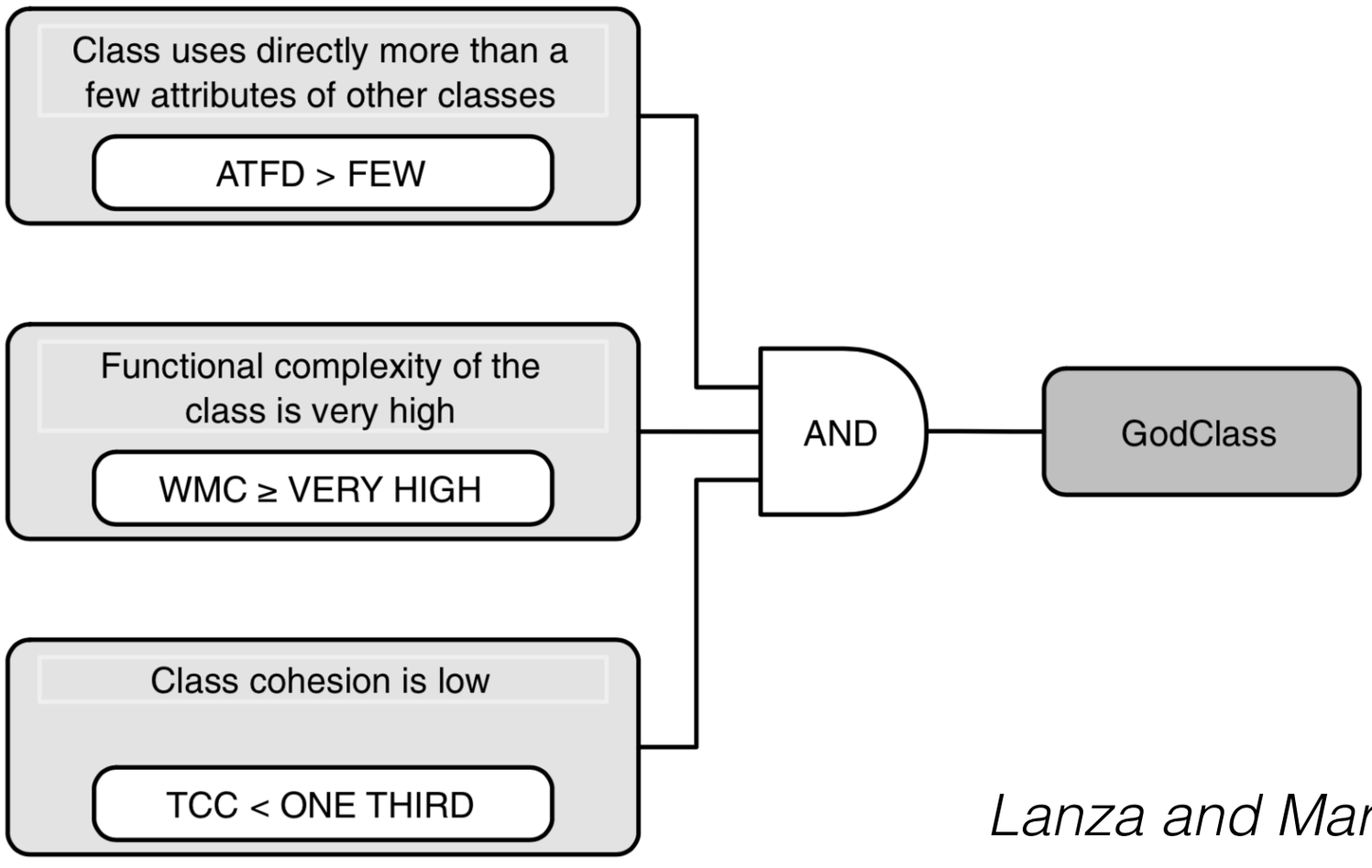
« *A method that seems more interested in a class other than the one it actually is in* »

Fowler (1999)

- Symptôme qu'une méthode est implémentée dans la mauvaise classe
- Beaucoup d'accès aux données de la classe enviée

Approches de détection

- Rule Card



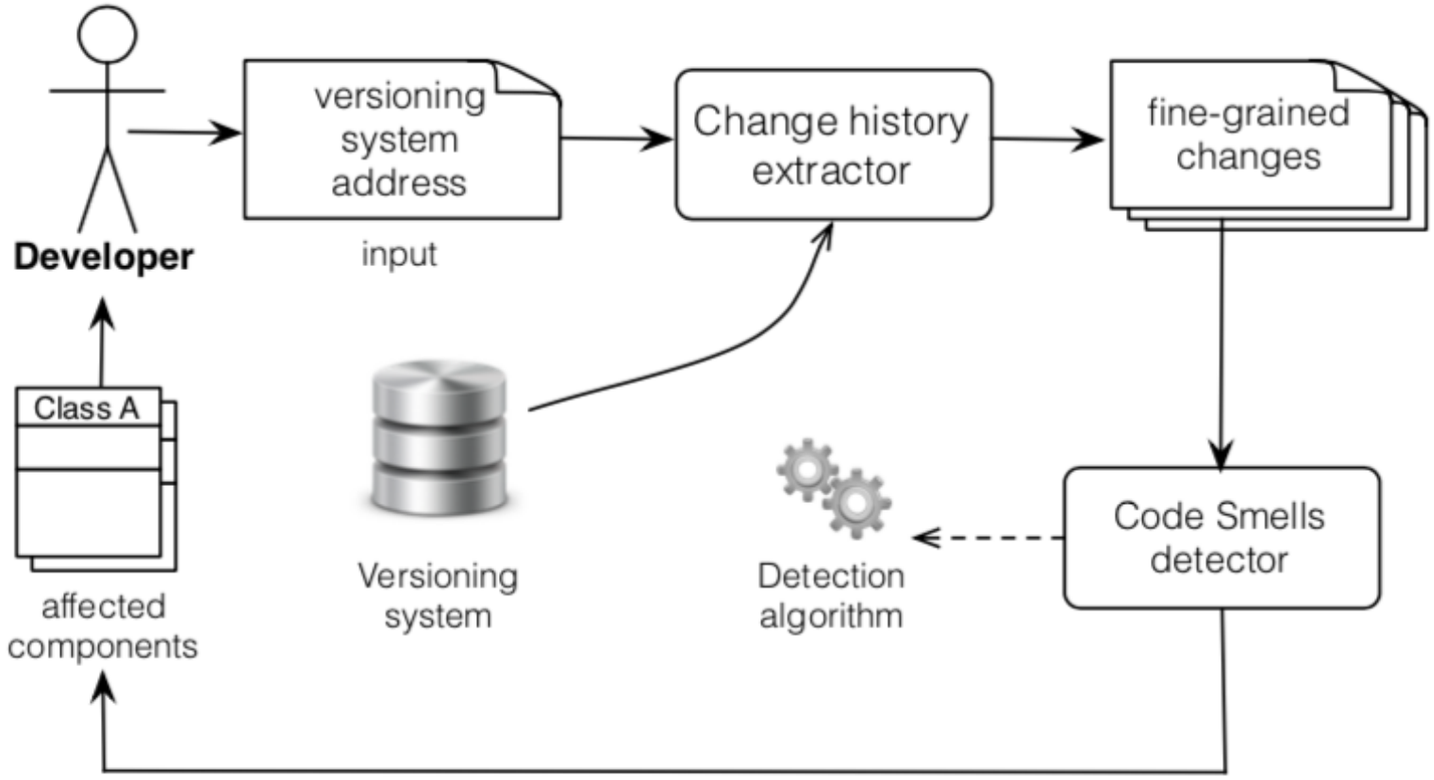
Lanza and Marinescu (2007)

Approches de détection

- Rule Card
- Opportunités de refactorisation

Approches de détection

- Rule Card
- Opportunités de refactorisation
- Information historique



Palomba et al. (2013)

Limites de ces approches

- Subjectivité des anti-patrons
- Métriques de détection définies arbitrairement
- Approches différentes, occurrences différentes
- Faible accord entre les différentes approches

Solution

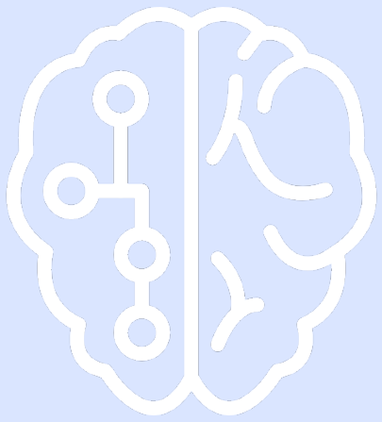
Tirer profit de l'**apprentissage profond** pour détecter les anti-patrons à partir de plusieurs sources d'information (**Structurelles** & **Historiques**)

- Approche statistique
- Pas besoin de définir explicitement les règles de détection
- Capable de traiter des données complexes

Qu'est ce que l'apprentissage profond?

Intelligence artificielle

Tout système accomplissant des tâches communément associées aux humains

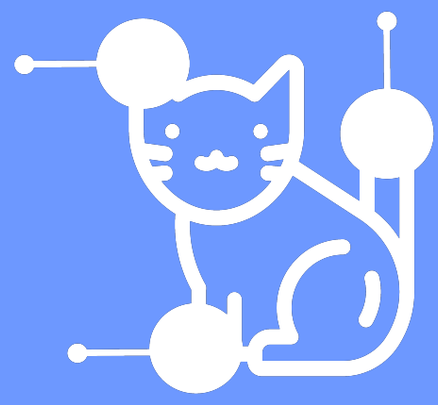


Apprentissage automatique

Apprendre l'accomplissement d'une tâche sans que celle-ci soit définie explicitement

Apprentissage profond

Apprendre les caractéristiques sous-jacentes des données en utilisant les réseaux de neurones



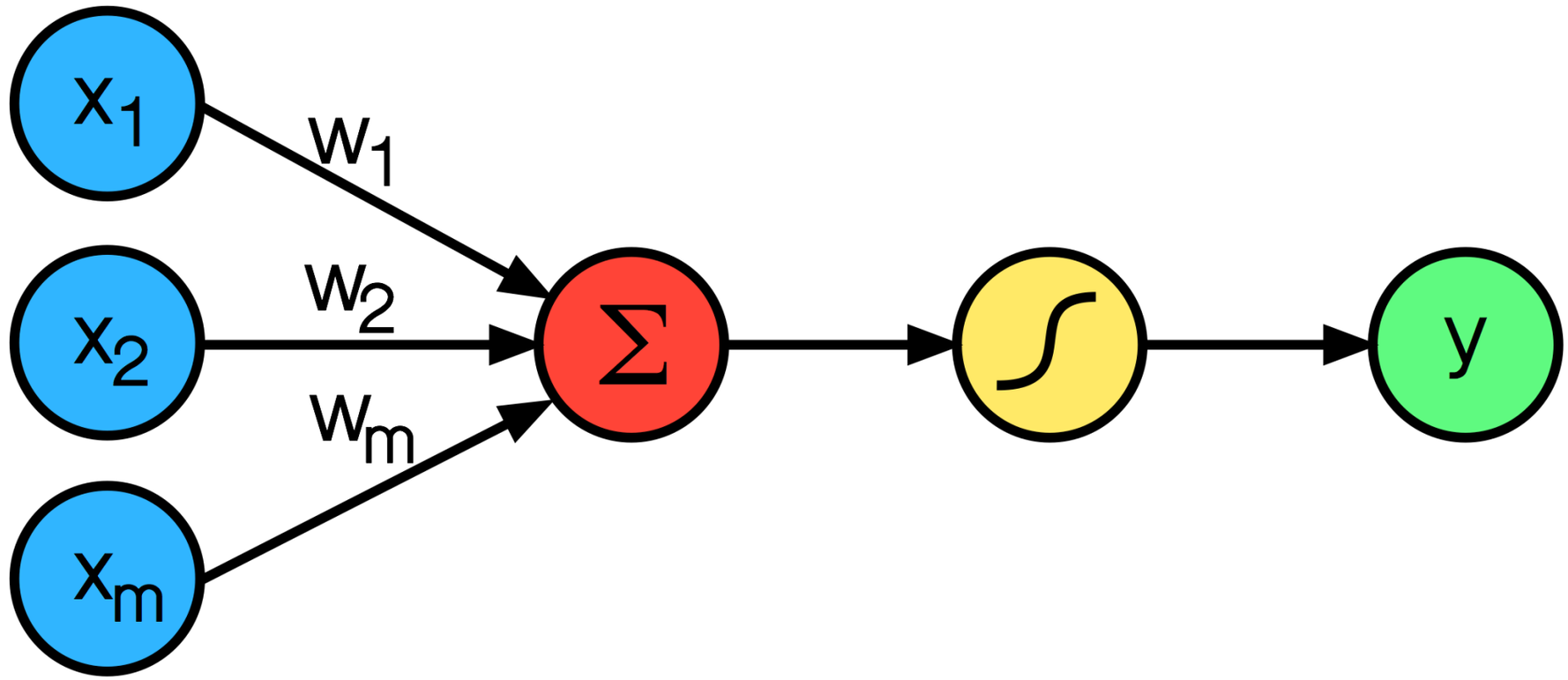
Réseaux de neurones

- Modèles probabilistes

$$\begin{bmatrix} x_1 \\ x_2 \\ \\ x_m \end{bmatrix} \longrightarrow \begin{bmatrix} P(C_1 \mid X) \\ P(C_2 \mid X) \\ \\ P(C_k \mid X) \end{bmatrix}$$

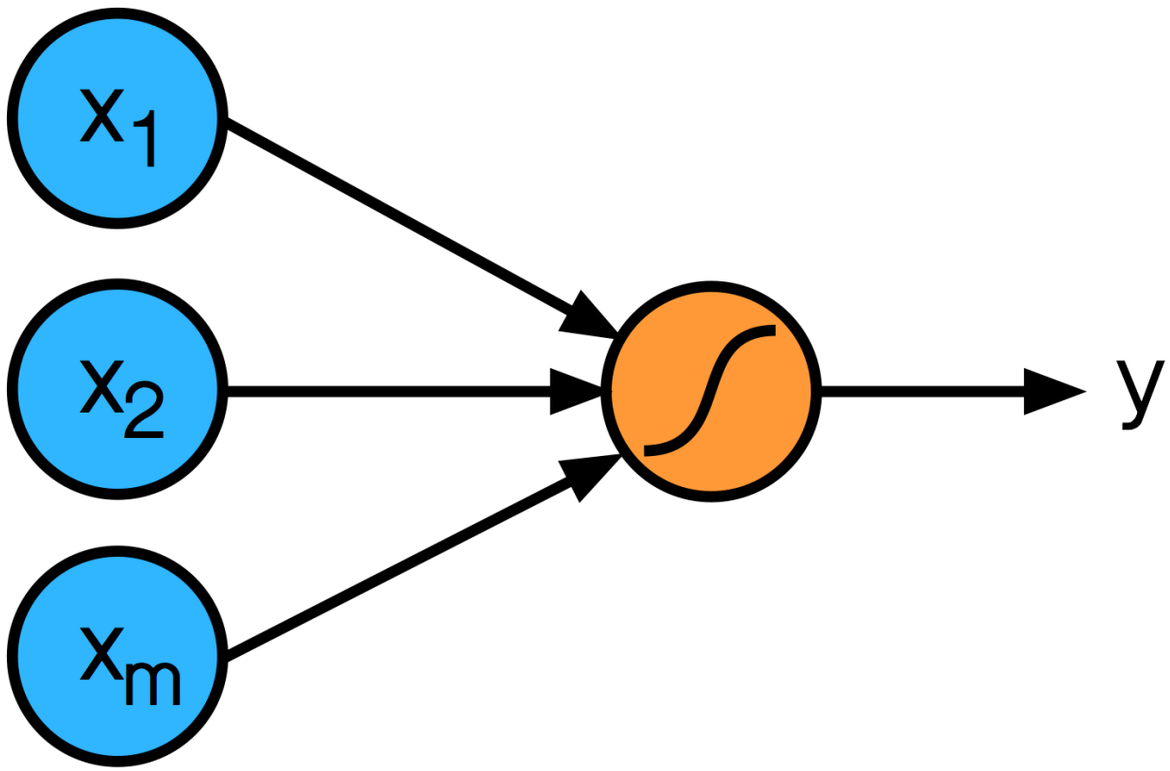
- Composés d'entités élémentaires appelées **neurones**

Le perceptron

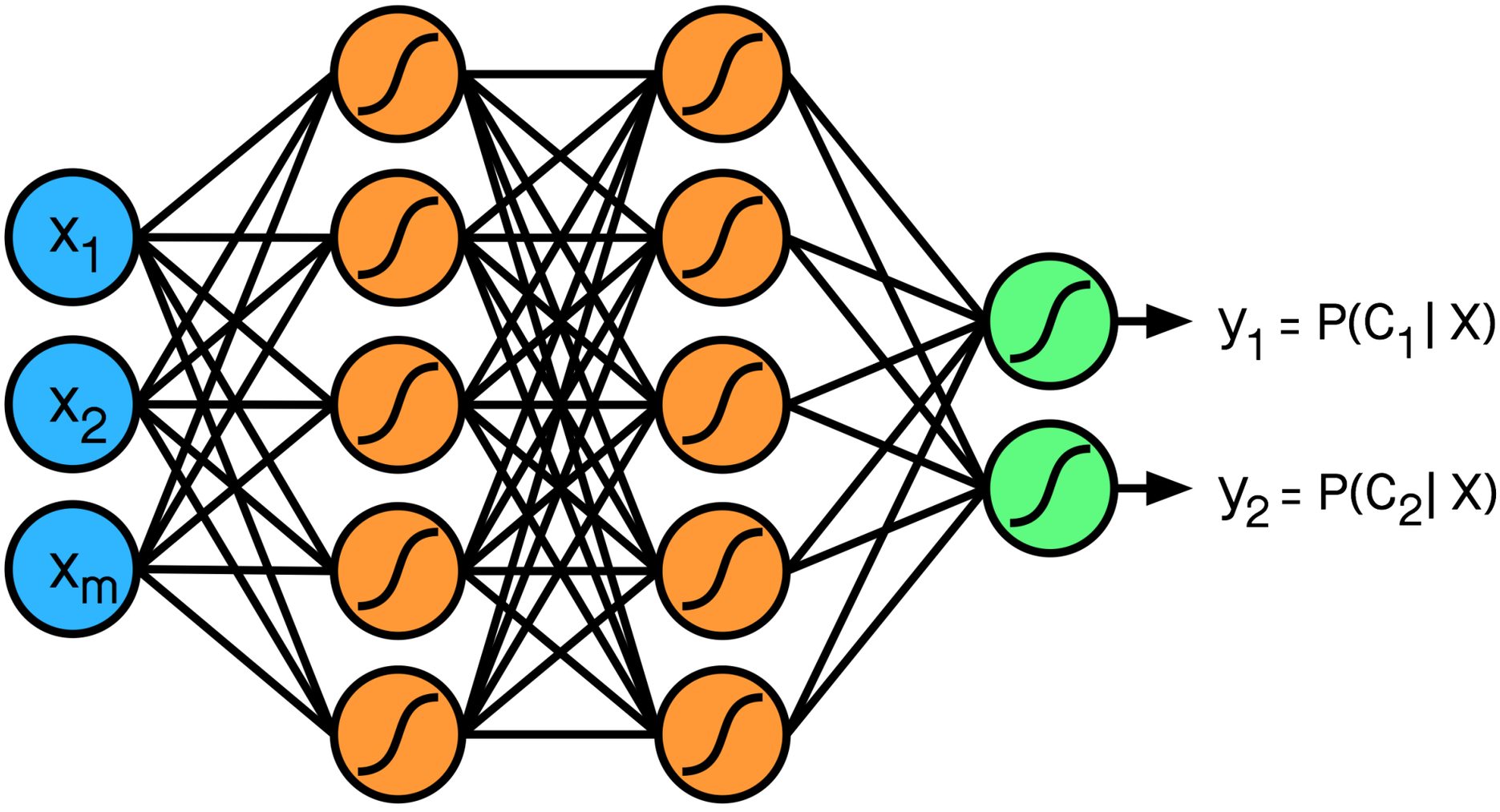


Entrées Poids Somme Activation Sortie

Le perceptron



Perceptron multi-couches (MLP)

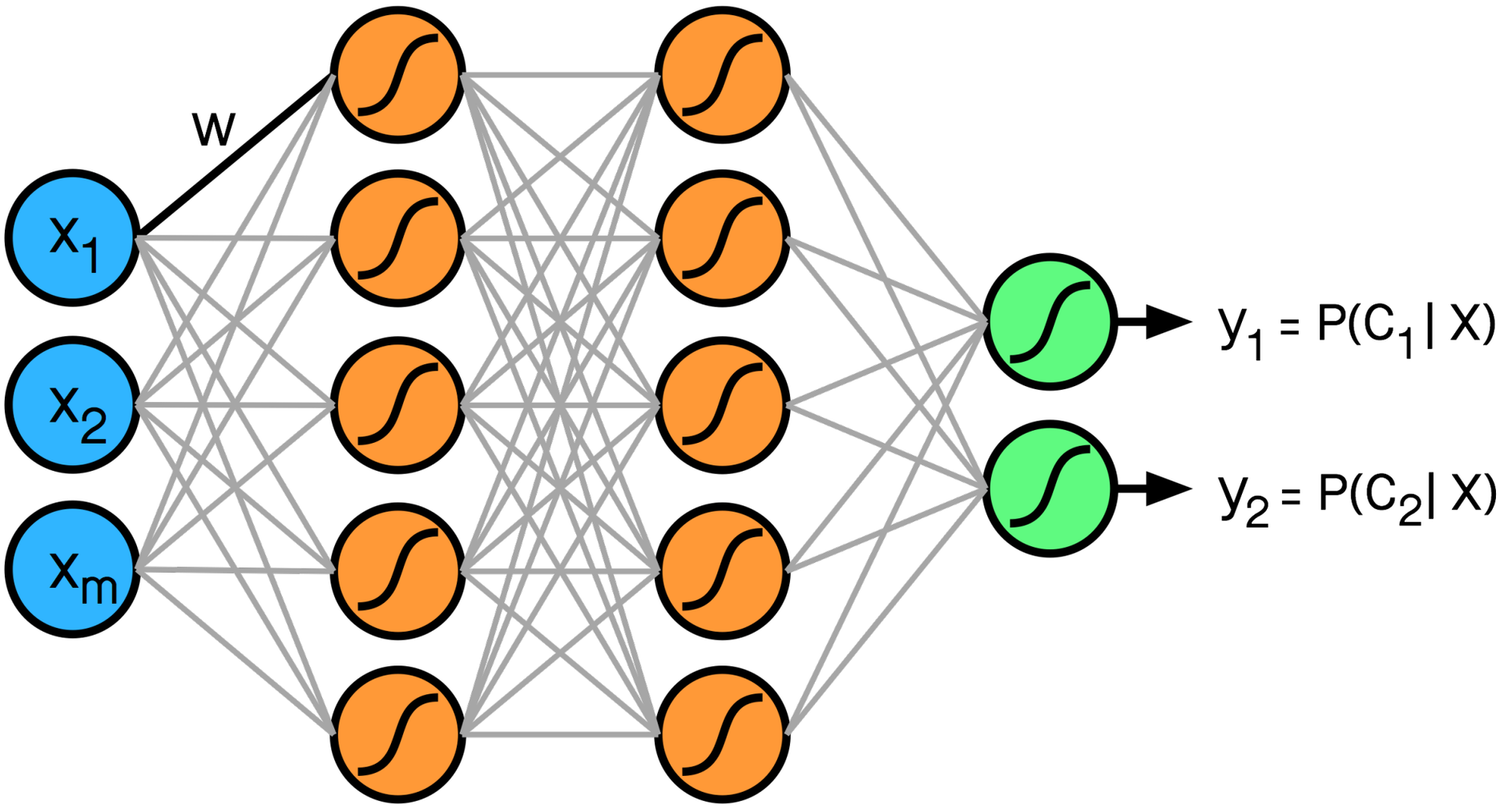


Couche d'entrée

Couches cachées

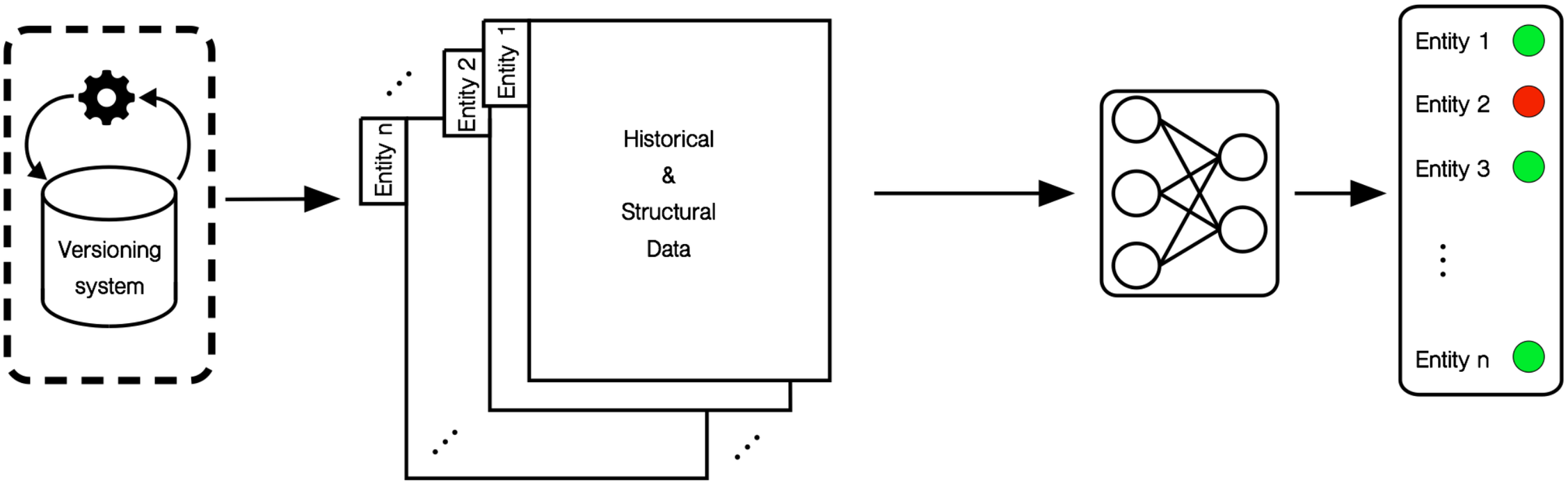
Couche de sortie

Apprentissage



Utilisation des réseaux de neurones pour la détection d’anti-patrons à partir des données Structurelles et Historiques

Utilisation des réseaux de neurones pour la détection d'anti-patrons à partir des données Structurelles et Historiques

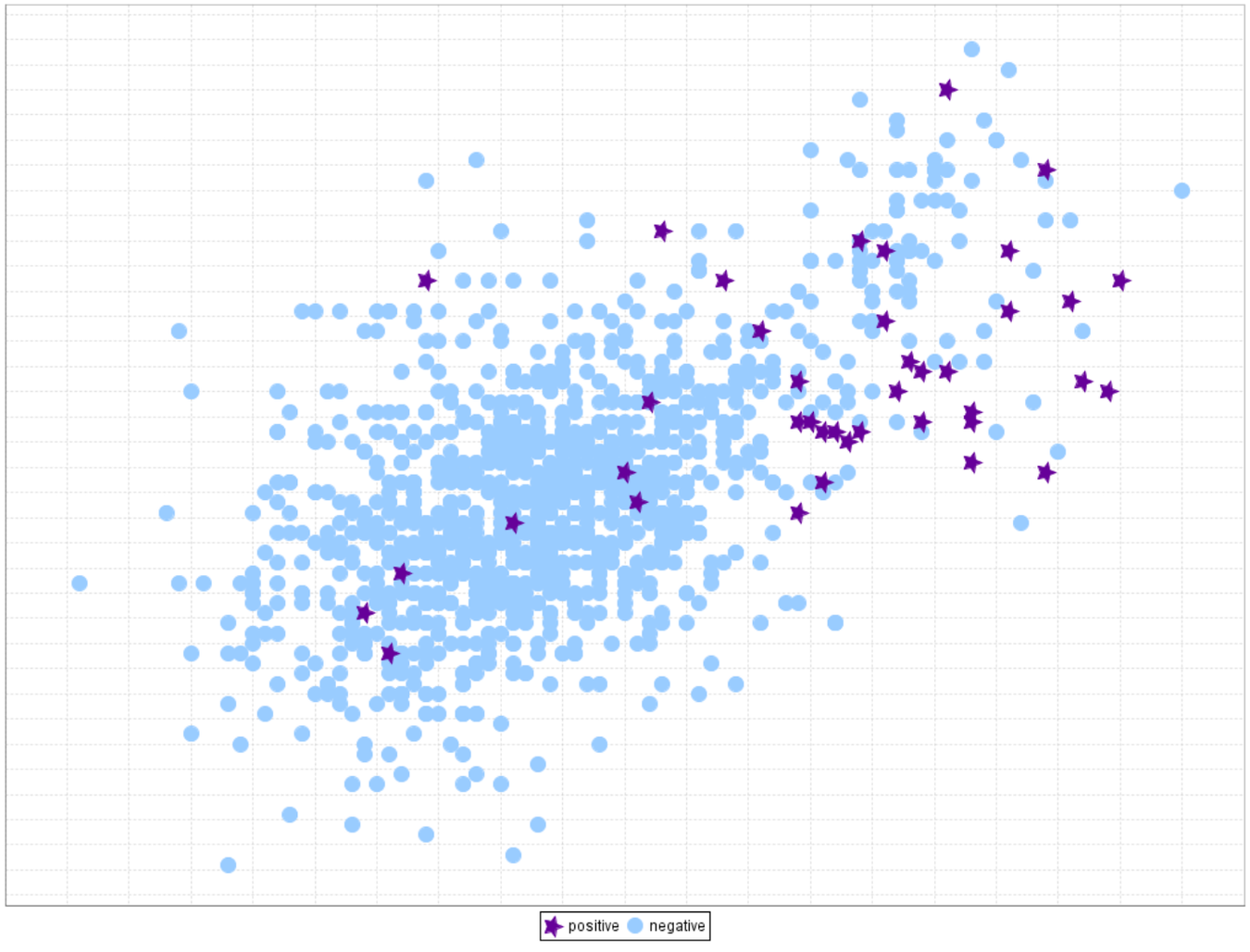


Oracle

- Nécessaire pour entrainer et évaluer nos approches
- Occurrences de God Class et Feature Envy dans un ensemble de systèmes
- Construit différemment pour God Class et Feature Envy

Systèmes	Taille	#God Class	#Feature Envy
Android Opt Telephony	192	11	18
Android Support	109	4	2
Apache Ant	694	7	25
Apache Tomcat	925	5	57
Apache Lucene	155	4	4
Apache Xerces	512	15	37
ArgoUML	1230	22	24
Jedit	423	5	22
Total	**4240**	**73**	**189**

Oracle: métriques d'évaluation



- Précision
- Rappel

F-mesure = 2x(PxR)/(P+R)

- Plusieurs approches de détection existent, basées sur différentes techniques et-ou sources d'information
- Des approches différentes identifient des occurrences différentes

- Plusieurs approches de détection existent, basées sur différentes techniques et-ou sources d'information
- Des approches différentes identifient des occurrences différentes

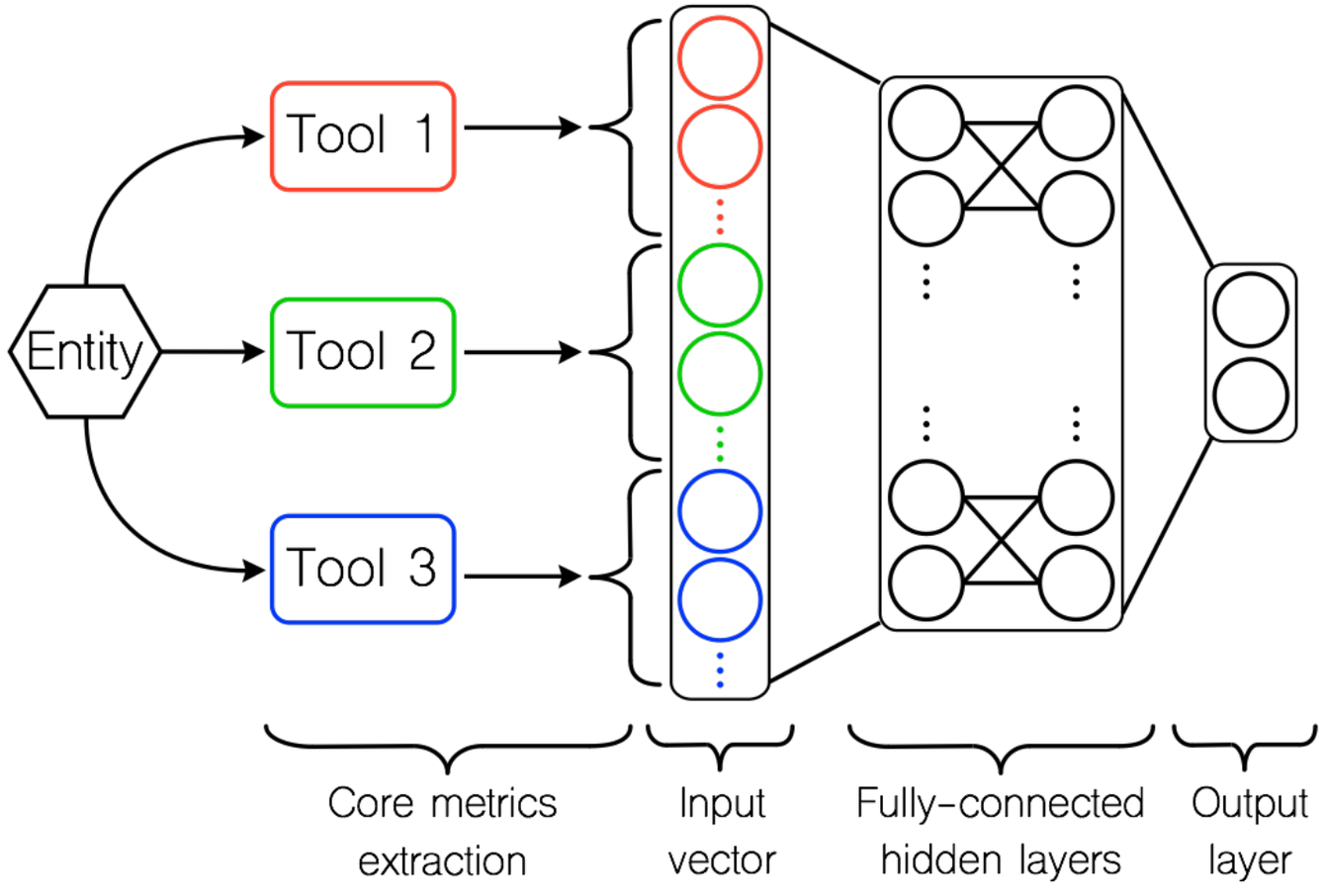
→ Combiner ces différentes approches pour améliorer les performances de détection

Choix des approches à agréger

Type d'approche	God Class	Feature Envy
Rule Card	DECOR	InCode
Refactoring	JDeodorant	JDeodorant
Historique	HIST	HIST

Agrégation

- Chaque approche à agréger est caractérisée par des **métriques**
- Utiliser ces métriques comme variables d'entrée à un réseau de neurones



Avantages

- Méthode d’agrégation plus complète qu’un simple vote
- Architecture peu complexe donc simple à entrainer
- Générer automatiquement des instances d’apprentissage pour des architectures plus complexes

- Ensemble d'entrainement: **5** systèmes
- Ensemble de test: **3** systèmes

RQ1: SMAD améliore-t-elle les performances des approches agrégées?

RQ2: Comment SMAD se compare à d'autres méthodes d'agrégation?

RQ3: Les occurrences détectées par SMAD peuvent-elles être utilisées pour entrainer d'autres modèles?

Moyenne des performances sur les trois systèmes

God Class

Approche	Precision	Rappel	F-mesure
DECOR	28 %	33 %	25 %
JDeodorant	8 %	48 %	11 %
HIST	41 %	38 %	38 %
SMAD	**74 %**	**63 %**	**66 %**

Feature Envy

Approche	Precision	Rappel	F-mesure
InCode	49 %	55 %	52 %
JDeodorant	59 %	47 %	50 %
HIST	4 %	4 %	4 %
SMAD	**74 %**	**67 %**	**70 %**

Moyenne des performances sur les trois systèmes

God Class

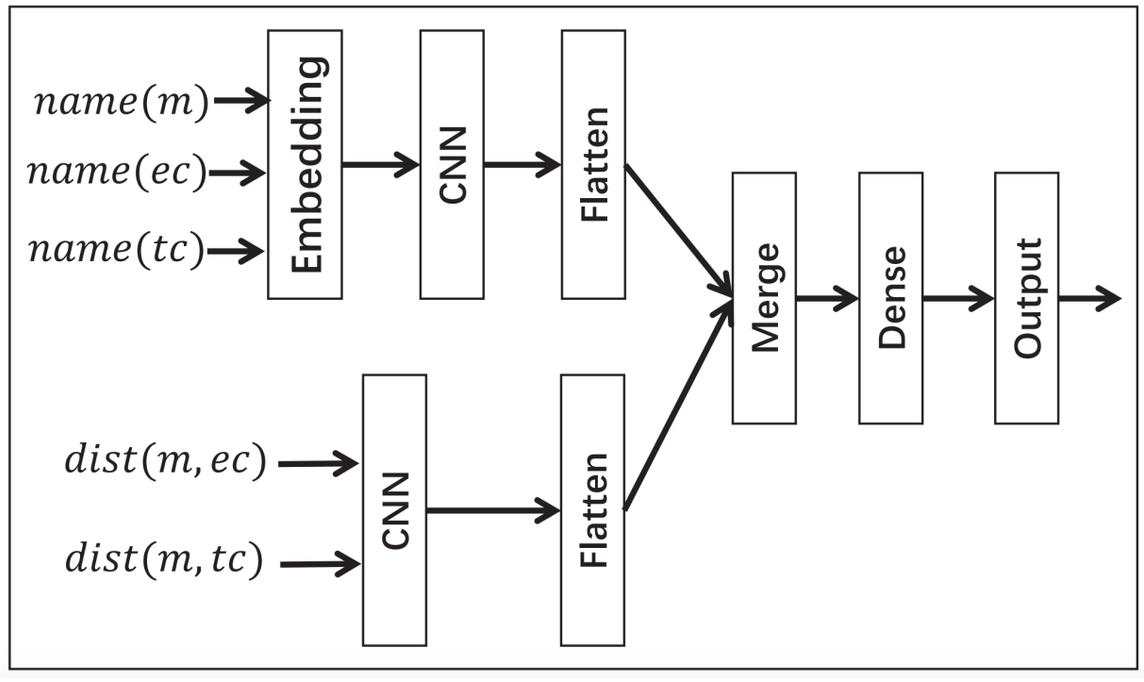
Approche	Precision	Rappel	F-mesure
Vote k=1	15 %	**78 %**	22 %
Vote k=2	71 %	28 %	31 %
Vote k=3	22 %	13 %	16 %
SMAD	**74 %**	63 %	**66 %**

Feature Envy

Approche	Precision	Rappel	F-mesure
Vote k=1	28 %	**100 %**	43 %
Vote k=2	52 %	7 %	12 %
Vote k=3	0 %	0 %	0 %
SMAD	**74 %**	67 %	**70 %**

RQ3: méthodologie

- Seulement pour Feature Envy
- Architecture sélectionnée: *Liu et al. (2018)*



- Entrainé sur 11 systèmes

Moyenne des performances sur les trois systèmes

Approche	Precision	Rappel	F-mesure
LIU_INJ	1 %	**94 %**	3 %
LIU_SMAD	**22 %**	13 %	**16 %**

- « Vrai » approche d’apprentissage profond
- Travaille avec des données complexes
- Traite l’information structurelle et historique simultanément

- « Vrai » approche d’apprentissage profond
- Travaille avec des données complexes
- Traite l’information structurelle et historique simultanément

→ Analyser l’évolution des métriques au cours du développement du logiciel

3 commits

```java
public class Dog {
    private String name;
    private int age;


    //Constructor
    public Dog(String name, int age) {
        this.name = name;
        this.age = age;
    }


    //Accesors
    public int getAge() {
        return this.age;
    }

    public setAge(int newAge) {
        this.age = newAge;
    }

    public String getName() {
        return this.name;
    }


    //Features
    public void bark() {
        System.out.println("WOOF WOOF!")
    }
}
```

NMD **NAD** **LOC**

CAME: exemple


```java
public class Dog {
    private String name;
    private int age;


    //Constructor
    public Dog(String name, int age) {
        this.name = name;
        this.age = age;
    }

    //Accesors
    public int getAge() {
        return this.age;
    }

    public setAge(int newAge) {
        this.age = newAge;
    }

    public String getName() {
        return this.name;
    }

    //Features
    public void bark() {
        System.out.println("WOOF WOOF!")
    }
}
```

	NMD	NAD	LOC
Commit 1	5		

CAME: exemple


```java
public class Dog {
    private String name;
    private int age;


    //Constructor
    public Dog(String name, int age) {
        this.name = name;
        this.age = age;
    }


    //Accesors
    public int getAge() {
        return this.age;
    }

    public setAge(int newAge) {
        this.age = newAge;
    }

    public String getName() {
        return this.name;
    }


    //Features
    public void bark() {
        System.out.println("WOOF WOOF!")
    }
}
```

	NMD	NAD	LOC
Commit 1	5	2	

CAME: exemple


```java
public class Dog {
    private String name;
    private int age;


    //Constructor
    public Dog(String name, int age) {
        this.name = name;
        this.age = age;
    }


    //Accesors
    public int getAge() {
        return this.age;
    }

    public setAge(int newAge) {
        this.age = newAge;
    }

    public String getName() {
        return this.name;
    }


    //Features
    public void bark() {
        System.out.println("WOOF WOOF!")
    }
}
```

	NMD	NAD	LOC
Commit 1	5	2	31

CAME: exemple


```java
public class Dog {
    private String name;
    private int age;


    //Constructor
    public Dog(String name, int age) {
        this.name = name;
        this.age = age;
    }


    //Accesors
    public int getAge() {
        return this.age;
    }

    public setAge(int newAge) {
        this.age = newAge;
    }

    public String getName() {
        return this.name;
    }
}
```

	NMD	NAD	LOC
Commit 1	5	2	31
Commit 2	4	2	25

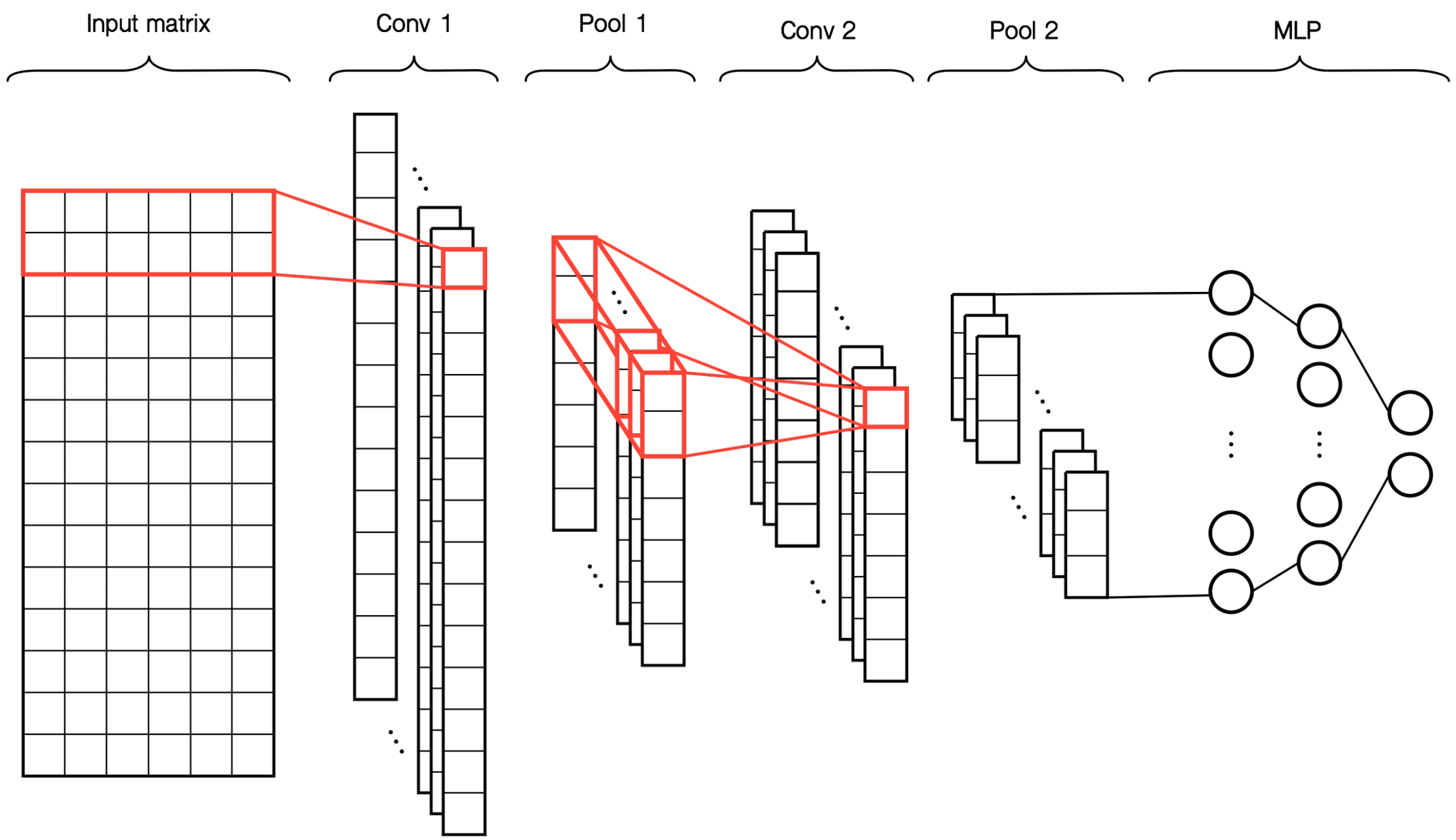
CAME: exemple


```java
public class Dog {
    private String name;


    //Constructor
    public Dog(String name) {
        this.name = name;
    }


    //Accessors
    public String getName() {
        return this.name;
    }
}
```

	NMD	NAD	LOC
Commit 1	5	2	31
Commit 2	4	2	25
Commit 3	2	1	15


Input matrix
Conv 1
Pool 1
Conv 2
Pool 2
MLP

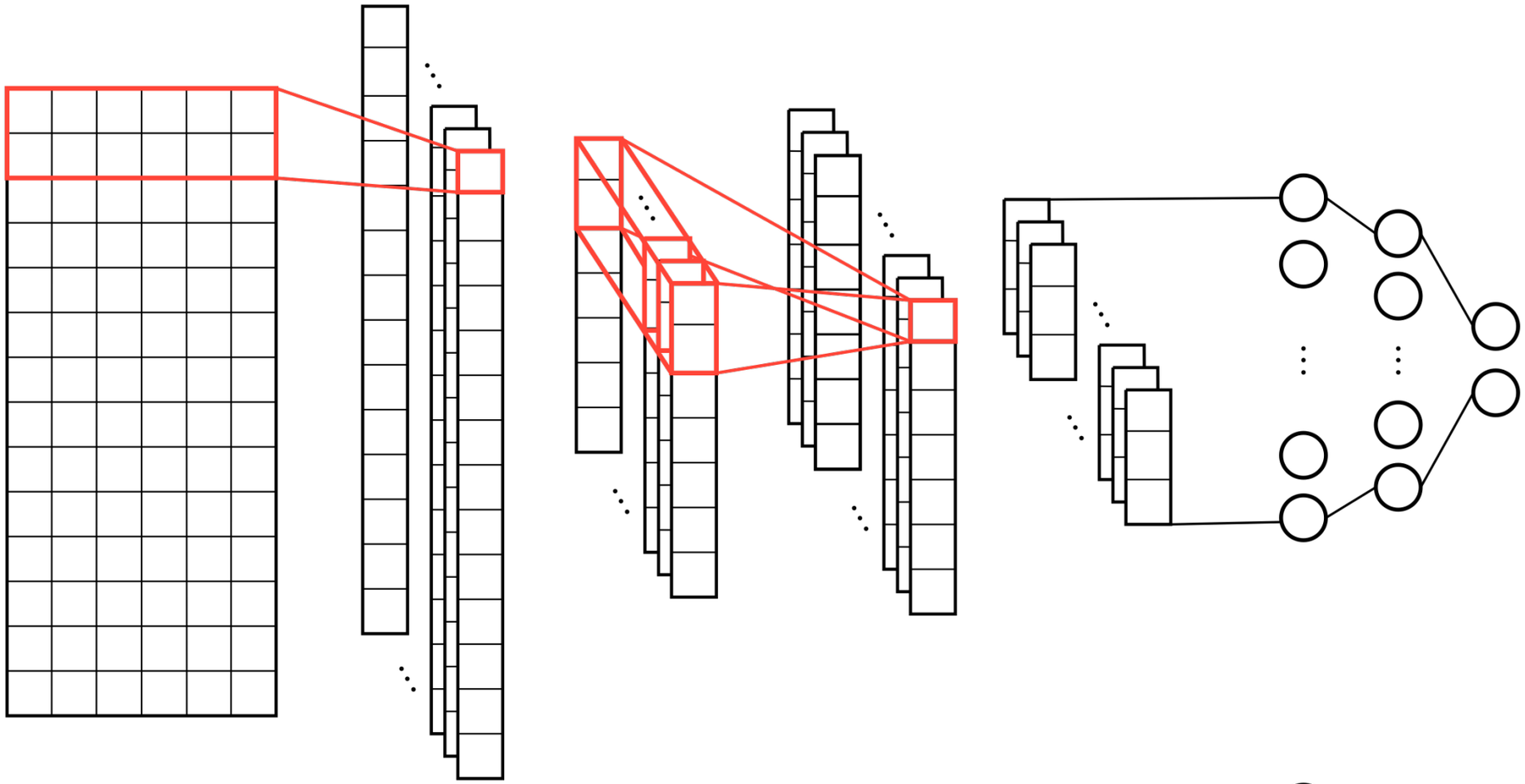
Étude

- Implémentation pour **God Class**
- Ensemble d’entrainement: **5** + **11** systèmes
- Ensemble de test: **3** systèmes

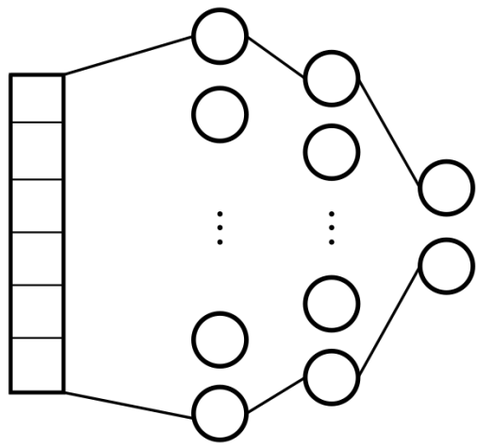
RQ1: L’historique des métriques améliore-t-il les performances?

RQ2: Comment CAME se compare aux approches de détection existantes?

CAME



MLP



Moyenne des performances sur les trois systèmes

God Class

Approche	Precision	Rappel	F-mesure
DECOR	28 %	33 %	25 %
JDeodorant	8 %	48 %	11 %
HIST	41 %	38 %	38 %
MLP	**73 %**	63 %	63 %
CAME	72 %	**78 %**	**72 %**

Conclusion

- Importance de la détection des anti-patrons dans les systèmes logiciels
- Limites des approches existantes
- Oracle
- Deux approches de détection basées sur le l'apprentissage profond reposant sur les données structurelles et historiques

SMAD:
- Agréger des outils de détection existants
- Générer des instances d'apprentissage

CAME:
- Analyser l'évolution des métriques logicielles

Travail futur

SMAD

- Experimenter d’autres architectures
- Comparer avec d’autres méthodes d’agrégation

CAME

- Étendre notre approche à Feature Envy
- Tirer profit des techniques de visualisation pour identifier les caractéristiques profondes et les causes premières des anti-patrons

Références

- W. J. Brown, R. C. Malveau, W. H. Brown, H. W. McCormick III, et T. J. Mowbray, *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*, 1st éd. John Wiley and Sons, March 1998.
- M. Fowler, *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA:Addison-Wesley, 1999.
- F. Palomba, G. Bavota, M. D. Penta, R. Oliveto, A. D. Lucia, et D. Poshyvanyk, *"Detecting bad smells in source code using change history information."* dans ASE, 2013, pp. 268–278.
- N. Moha, Y. Guéhéneuc, D. Laurence, et L. M. Anne-Franccoise, *"Decor: A method for the specification and detection of code and design smells"*, IEEE Transactions on Software Engineering (TSE), vol. 36, no. 1, pp. 20–36, 2010.
- M. Fokaefs, N. Tsantalis, E. Stroulia, et A. Chatzigeorgiou, "Jdeodorant: identification and application of extract class refactorings", dans Software Engineering (ICSE), 2011 33rd International Conference on. IEEE, 2011, pp. 1037–1039.
- M. Lanza et R. Marinescu, *Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems*. Springer Science & Business Media, 2007.