netPurpose

This document summarizes my findings on different ways to extract features from source code and find similarity between 2 source codes.

Research papers

1. Document vectorizer: research a method to calculate document vectors by first finding word vectors. Compare this method with results obtained using paragraph2vec method. The method should not require expensive inference technique (like paragraph2vec method has).
2. Parse tree kernel method (paper): there are many kernels to compare 2 text documents. There are several variants of these kernels so that they can also be used to compare 2 program source codes.
3. Checkout stanford Moss.
4. Comparison of various source code similarity methods.
5. Detecting Source Code Similarity Using Code Abstraction.
6. Automatic algorithm recognition based on programming schemas and beacons.
7. Assessing roles of variables using program analysis.
8. A Similarity Ranking of Python Programs
9. Overcode

My findings

Automatic algorithm recognition based on programming schemas and beacons:

- The idea explored in the paper is to identify algorithm from written programs directly. To perform this analysis they've used idea of schemas and beacons.
- First they try to identify schemas from program code. Schema is general algorithmic structure of the program. They have some hand crafted schemas and they compare the program to find out which schema is most appropriate. If it is not possible to make such decision then they use raw program otherwise they use the identified schema for further analysis.
- Beacons are some characteristic attribute of program code. It includes numerical measures such as halstead's metrics, cyclomatic complexity, number of assignment statements etc..
- Beacons also include roles of variables information. They first identify all variables used in program and then they identify roles of each variable. Using these beacons they obtain a feature vector which contains all numerical characteristics and roles of variables (like how many variables of each role does a program have) information.

- From these feature vector they try to classify each program using decision tree classifier to identify their algorithm.

Assessing roles of variables using program analysis:

- Main purpose of this research paper is to analyse roles played by different variables in the program.
- They perform statistical analysis of the program. The entire procedure is as below,
- First they generate a source code map of the program. The source code map contains all the statements present in root block. If a statement is a control statement then it is linked with the body of that control statement. So basically we get a tree like structure of all statements in program where each node contains all block level statements and control statements are linked to child nodes which contain all statements in that control block.
- Once we have source code map we can get a program slice of a variable. So before we get a program slice we identify all the variables in the program. According to paper, *"A slice of a program with respect to a variable is the program with all statements which cannot impact on that variable removed."*
- After getting program slice of variable they try to find assignment, usage, conditional usage and other statements. They classify all statements in program slice into one or more of those four categories.
- Then based on these four categories of statements they make 21 propositions (from A to U). They are called 21 rules. Each rule can have true or false for a particular variable. For ex, whether a variable is used in loop, whether variable is assigned in loop in used outside loop, whether variable is incremented in loop, etc…
- From these 21 propositions they can identify 11 distinct roles which are boolean functions of above 21 rules. For ex, the failure condition for *most recent holder* is Y = (F ∨ ¬A ∨ (Q ∧ R) ∨ (Q ∧ X) ∨ M ∨ K ∨ I ∨ J ∨ S ∨ H ∨ L ∨ (¬G ∧ D ∧ ¬P) ∨(¬T ∧ ¬U) ∨ (¬B ∧ (¬C ∨ D ∨ E))) (if this boolean function is true then variable is not a most recent holder).
- The research and source code are all publicly available. They have Java implementation of this method available as a BlueJ IDE extension. Download the zip inside it contains one more reserach paper which is more detailed and zip contains all java files used for this analysis.

Stanford MOSS

- MOSS is an online web based service which checks plagiarized code. The purpose of MOSS is to have a plagiarism tool for education. This paper describes the algorithm being used in MOSS.
- At the heart of MOSS is **winnowing** algorithm. Which is based on document fingerprint. Basic approach is to generate a document fingerprint and then compare it with some other documents fingerprint. Using this fingerprints we can measure similarity between two documents.

- The winnowing algorithm is based on concept of k-gram hash. That is after removing unnecessary information from a document (like whitespaces). You generate all possible strings of length K. (So if your document has L characters then there are L - K + 1 possible strings).
- Then using a hash function convert these strings (of length K) to numeric value. The hash function should assign values uniformly with less probability of collision.
- Winnowing uses concept of windows. For this it uses 2 values, namely, T and K. T is the threshold value. That is if there is a substring match at least as long as the guarantee threshold, T, then this match is detected. Thus T decides the minimum length of substring that user wants to detect.
- K is a noise threshold. That is we never detect strings whose length is below the noise threshold.
- This values are supplied by user and depending on values of T and K, the size of window is decided (W = T - K + 1).
- Therefore now we generate hash values using K-grams (with K being noise threshold). Then we create all possible windows of size W. From all these windows we select minimum numeric value for digital fingerprint. Each value is selected only once.
- So even let say first window is [110, 20, 30, 17, 50] and second (consecutive) window is [20, 30, 17, 50, 78]. Then we select 17 from first window and second window will also select 17. But since it is same 17 we select that 17 only once.
- Window size W > T - K allows us to match all strings with length T or more. (Since there will be T - K + 1 hashes of substring of length T, this number is same as window size)
- This way all the hash values which we selected make up digital fingerprint of the document. We can extract fingerprints from 2 document and use any of the set similarity metrics to measure the similarity between them.
- The advantage of MOSS is in query , too. If you want to query some document against stored fingerprints then you can select windows size W' > W (original size using which stored fingerprints were generated) and still be able to measure similarity. Since F(W') is subset of F(W). (F(W) being fingerprint chosen for the document using winnowing and windows size W).

A comparison of various source code similarity methods

- The research paper compares various source code similarity measurement methods to find out accuracy of these methods for detecting code plagiarism.
- It compares AST, CFG, W-shingling and MOSS (Winnowing). There were 5 types of attack cases. Each attack case had two programs with some alteration in them.
    - Comment alteration: the two input programs were same but had different comments or same comments but at different position.
    - Whitespace padding: same programs but second one was padded with whitespace.
    - Identifier alteration: same programs but with different names of variables.

- Code reordering: same program but function definitions were differently ordered.
- Algebraic expression alteration: same program with expression altered slightly. For e.g. ‘4 + x’ and ‘x + 4’, ‘a * (b + c)’ and ‘a * b + a * c’.

Result of different methods for similarity measurement.

	AST	CFG	Lev (src)	Lev (IR)	W-shingling (src)	W-shingling (IR)	MOSS
Comment alteration	1.0	1.0	1.0	1.0	1.0	1.0	0.99
Whitespace padding	1.0	1.0	0.942	1.0	.0646	1.0	0.99
Identifier alteration	1.0	1.0	0.654	0.9994	0.0655	0.979	0.99
Code reordering	0.9299	1.0	0.3425	0.4616	0.679	0.971	0.82
Algebraic expression	0.611	0.6	0.8926	0.5229	0.546	0.746	0.68
root.c	0.1832	0.0573	0.2	0.224	0.0	0.1754	0.0

(root.c is entirely different program)

- AST is abstract syntax tree. To measure similarity using AST they’ve used RTED tree edit distance algorithm.
- LEV is levenshtein distance between programs. Src compares source codes directly while IR compares IR code generated by LLVM compiler.
- CFG is control flow graph. To find similarity using CFG first they generate CFG from program and then generate its adjacency matrix. Then serialize this matrix into a binary string and then compare leviathan distance between two string of two programs.
- W-shingling is a document fingerprinting algorithm. It is similar to winnowing but they differ in one aspect. Winnowing only takes subset of all k-gram hashes as document’s fingerprint but W-shingling takes all the hashes into fingerprint.
- MOSS is an online web based utility to check plagiarism between two programs but they use winnowing as main component.

Overcode

- Overcode is an program source code visualizer. This tool takes a bunch of programs and arranges them according to their similarity and based on that they allow analysis of programs. Their analysis pipeline is as follows:
- First they sort out all the source codes and remove unnecessary parts such as comments.

- Second they execute this codes and extract variable sequences from this codes. Variable sequences are sequence of values that a variable holds during execution of program.
- Then they check these variables' sequences of 2 or more programs and identify common variables which have same variable sequence. They rename all common variables and also identify unique variables and name them differently.
- After renaming they identify program stacks. A program stack is set of program which are considered in same group. This identification is done on modified source codes and by comparing source code directly. If codes have same outcome as execution they are considered in same stack.
- That's how they identify dissimilar programs and allows to conduct variety of analysis on these source codes.

Detecting Source Code Similarity Using Code Abstraction

- In this paper goal is to identify similarity between programs by making abstraction of code.
- First they abstract the code by remove unnecessary information which according to them play no or negligible part in similarity checking. Here they remove all the statements which are not control structure statements, all function calls (but they keep standard library function calls), all variables, all constants, all literals. So in the end they obtain a skeleton of the program code. For e.g. see image below, which describes this skeleton structure obtained using code abstraction.

Origianl Source Code

```
1 /* Average of input digit */
2 #include <stdio.h>
3 int main()
4 {
5   int n, a[1], *ptr;
6   int i, sum;
7   printf("The number of input?");
8   scanf("%d", &n);
9   if (n>0) {
10    for (i=0; i<n; i++) {
11      printf("Input digit:");
12      scanf("%d", &a[i]);
13    }
14    ptr = &a[0];
15    for (i=0, sum=0; i<n; i++)
16      sum += *(ptr+i);
17    printf("Average=%d\n", sum/n);
18  }
19  else
20    printf("Try again.\n");
21  return 0;
22 }
```

Abstracted Source Code

```
1 #include <stdio.h>
2 int main()
3 {
4   printf("");
5   scanf("", (& n));
6   if (n>0)
7   {
8     for (;;)
9     {
10      printf("");
11      scanf("", (& a[i]));
12    }
13    for (;;)
14    {
15    }
16    printf("", (sum/n));
17  }
18  else
19  {
20    printf("");
21  }
22 }
```

Figure 3: Example of original source code and abstracted source code

- Then they identify similarity between two or more programs by finding similarity between the skeletons of program which is obtained using Moss utility.
- So code similarity analysis pipeline is to make code abstract and find similarity using Moss.

Dataset

Link: https://github.com/prasanna08/code-classifier-dataset

Description:

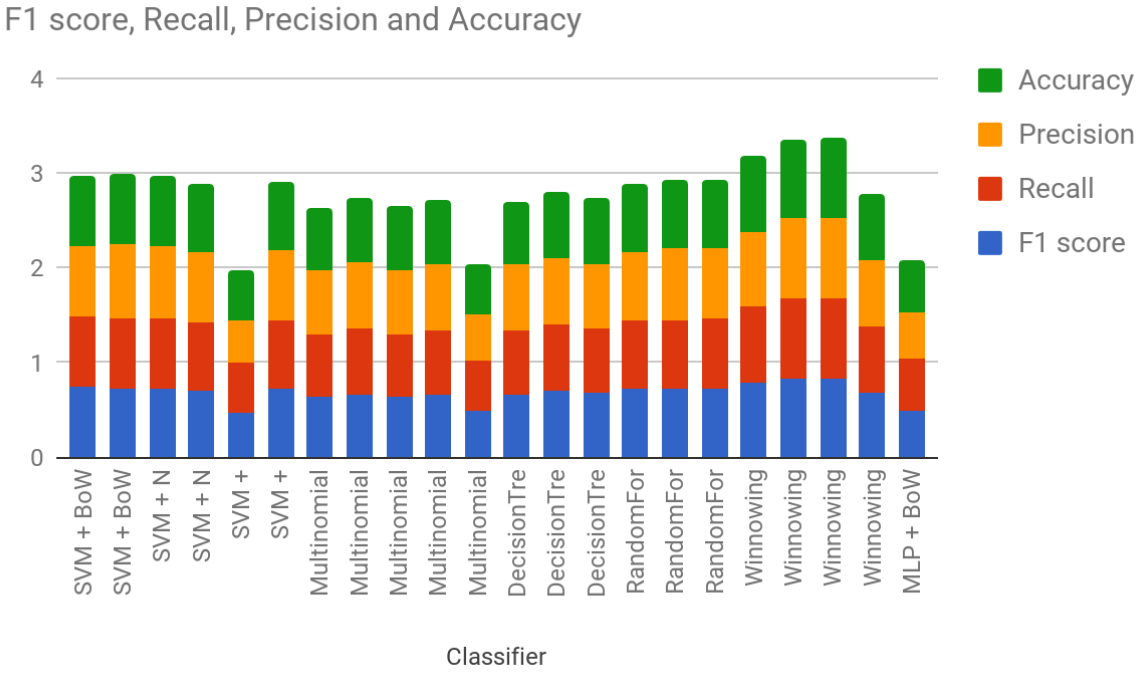
- See README.MD file in above repo.

Spreadsheets:

- Feedback class description: contains class ID and their corresponding feedback statements.
- Execd dataset with classes: contains programs in ‘execd_dataset.json’ with each program having unique ID and feedback class assigned to it.
- Compiled dataset with classes: contains programs in ‘compiled_dataset.json’ with each program having unique ID and feedback class assigned to it.

Results

- I have been testing different classifiers on the tagged dataset. The summary of



classification report of these classifiers can be found here.

- I have also uploaded code of these classifiers on my GitHub repository.