AI based Malware detection approach for KISA Data challenge 2018

Dec 1st, 2018

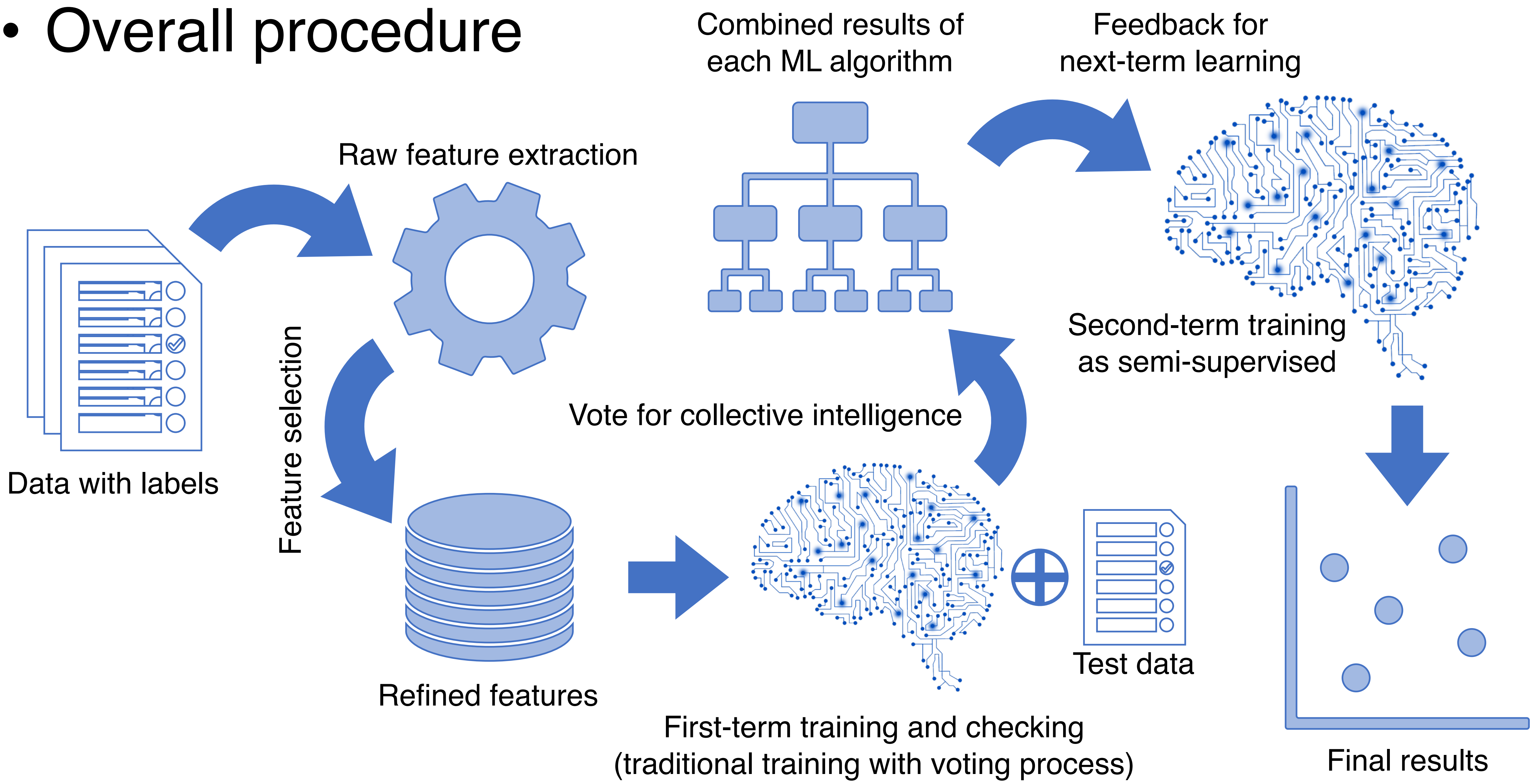
Hyunsoo Kim

aitch25@gmail.com

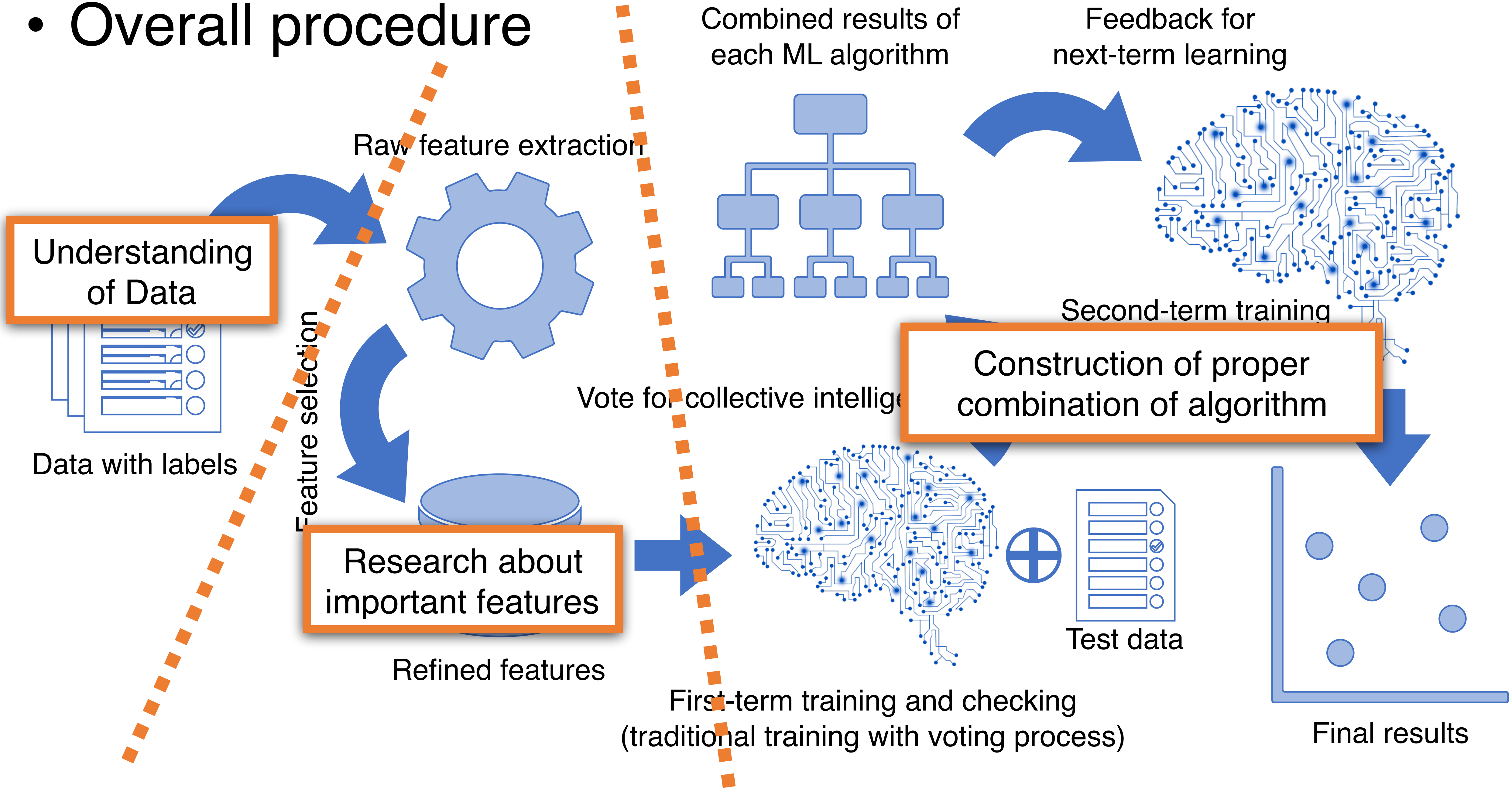
Assistant researcher at F1 Security

- # Index
  - Overall Procedure
  - Research works
    - Data analysis
  - Feature selection
    - Unavailable features
    - Importance features
  - Detection algorithms
    - Traditional approaches
    - How we improve it with feedback approach

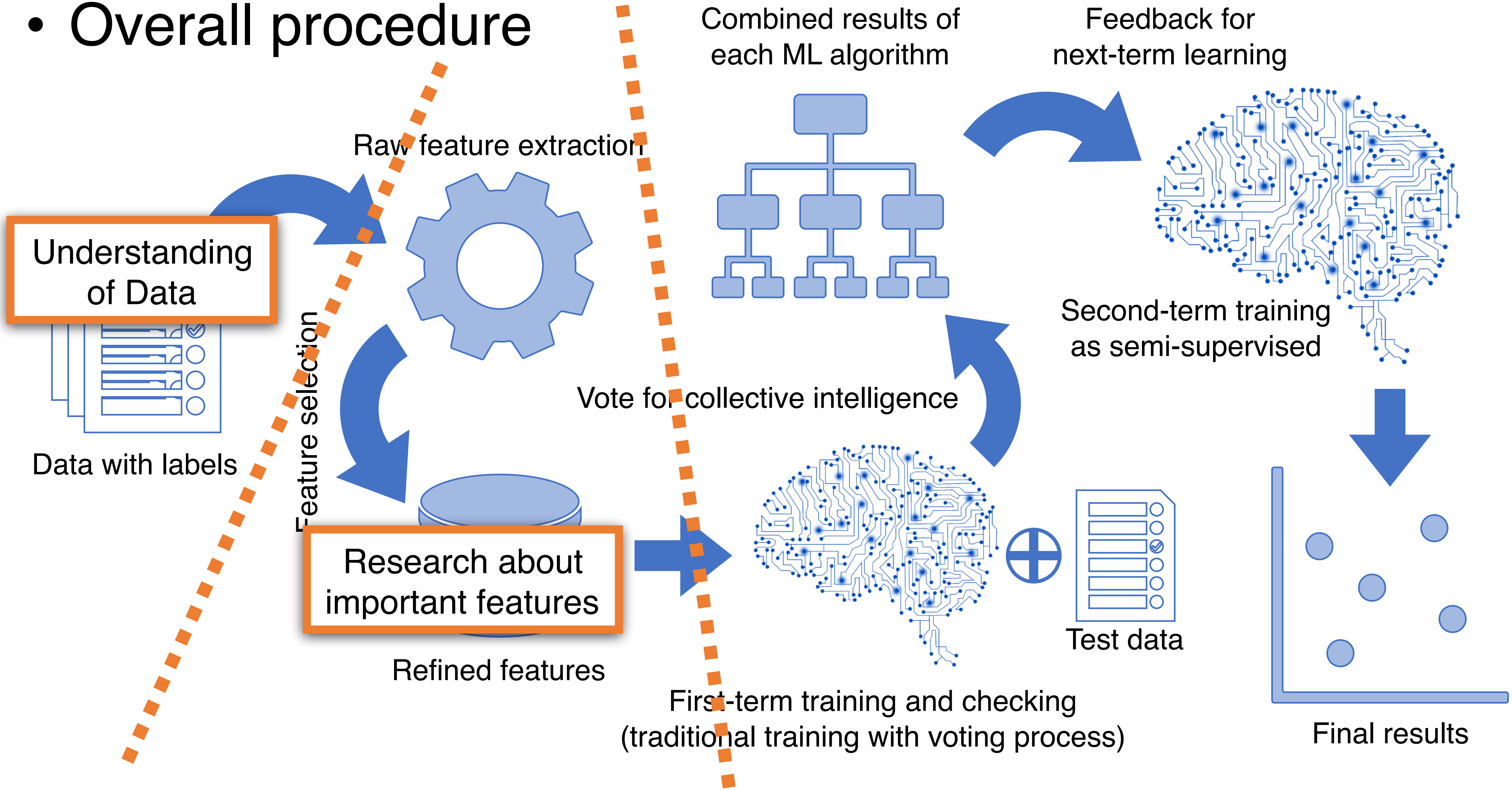
- # Overall procedure



- Overall procedure

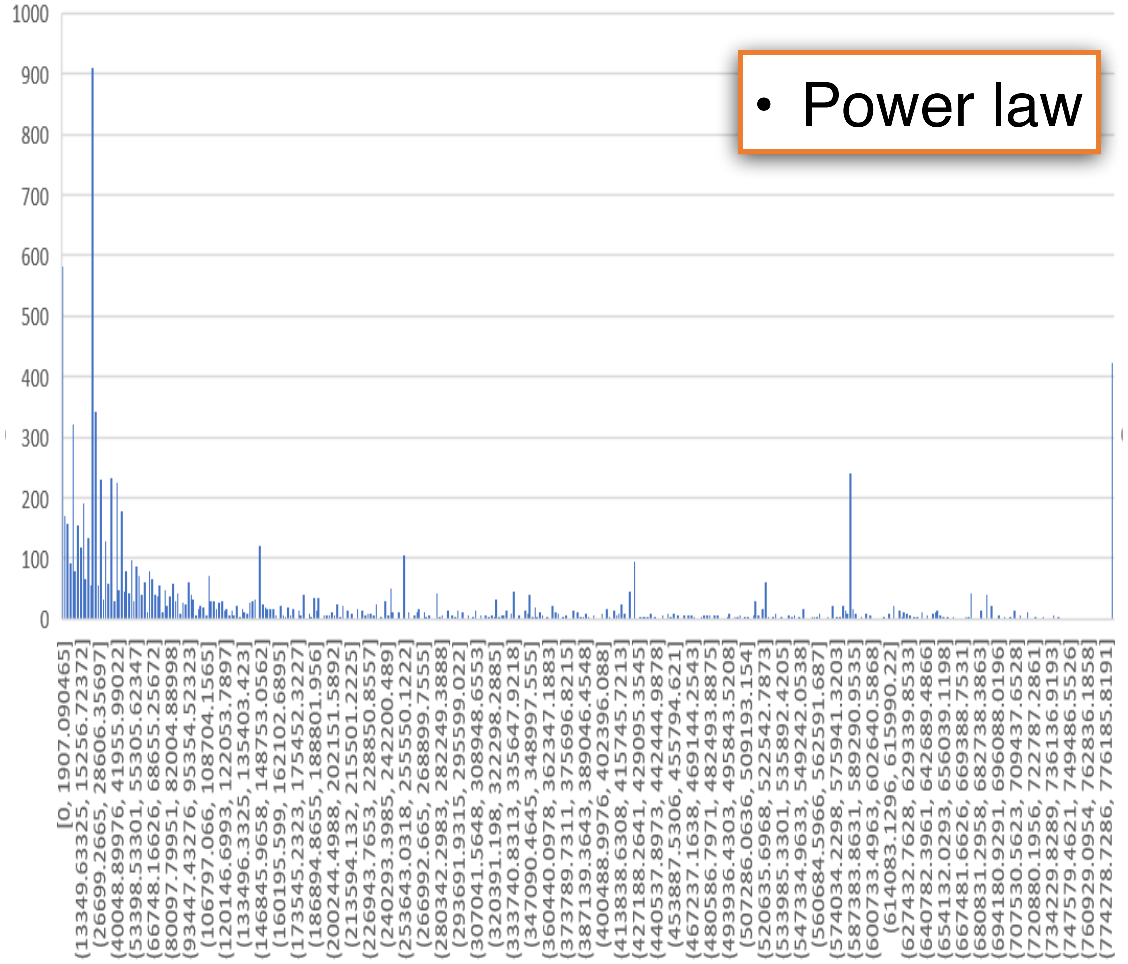


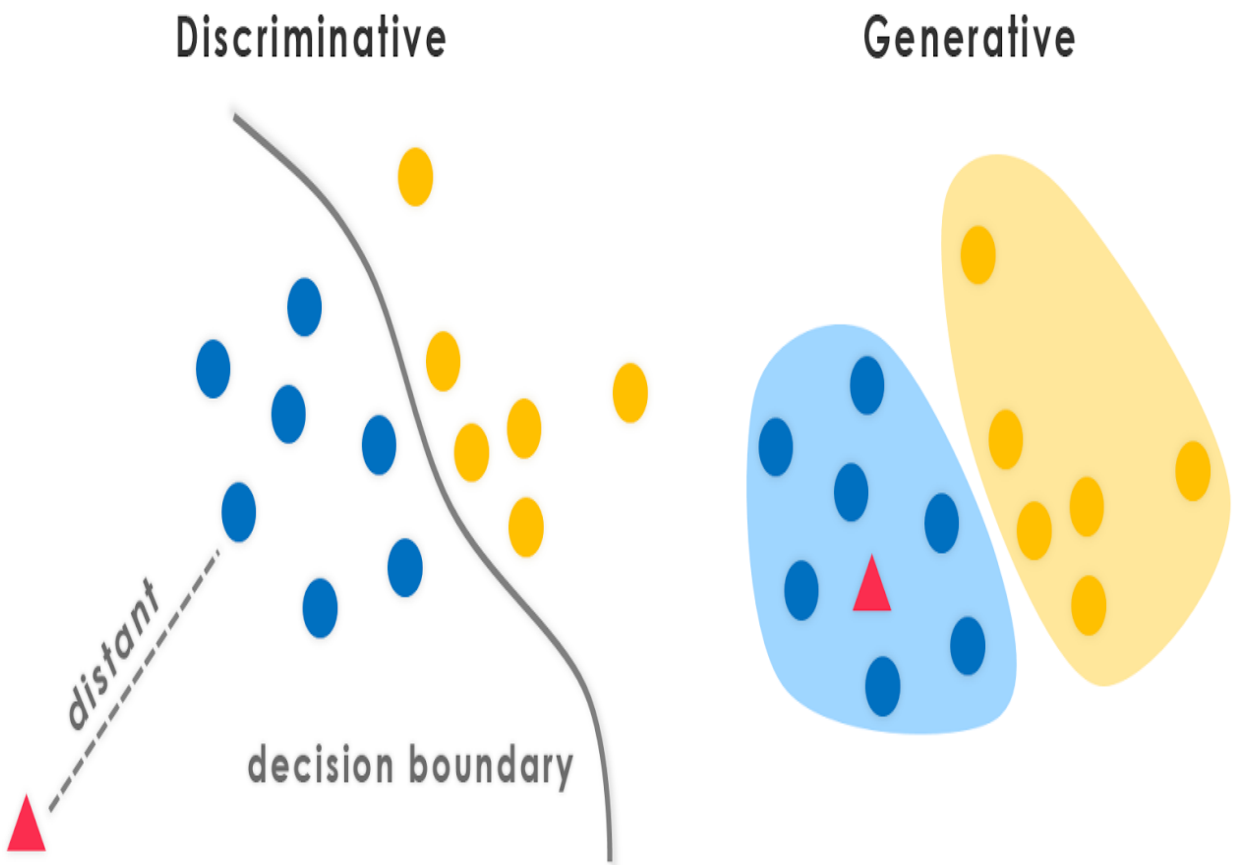
- Overall procedure



- Research works

- Data analysis
  - However, dataset has ambiguous forms..





- Feature selection

- Collecting features as many as possible.. (S: static // D: dynamic feats)
  - (S) Feature list: 86 feature set (extracted by pefile API)
    - ex) Size of code, Address of entry point, etc..

  - (S) 256-gram of binary file [2][3]
  - (S) TFIDF of strings (with readability checker) [4]
  - (S) TFIDF of imported DLL
  - (S) Image representation [2]
  - (D) Bi or Tri-gram of API Sequence (using Cuckoo and Virus total)

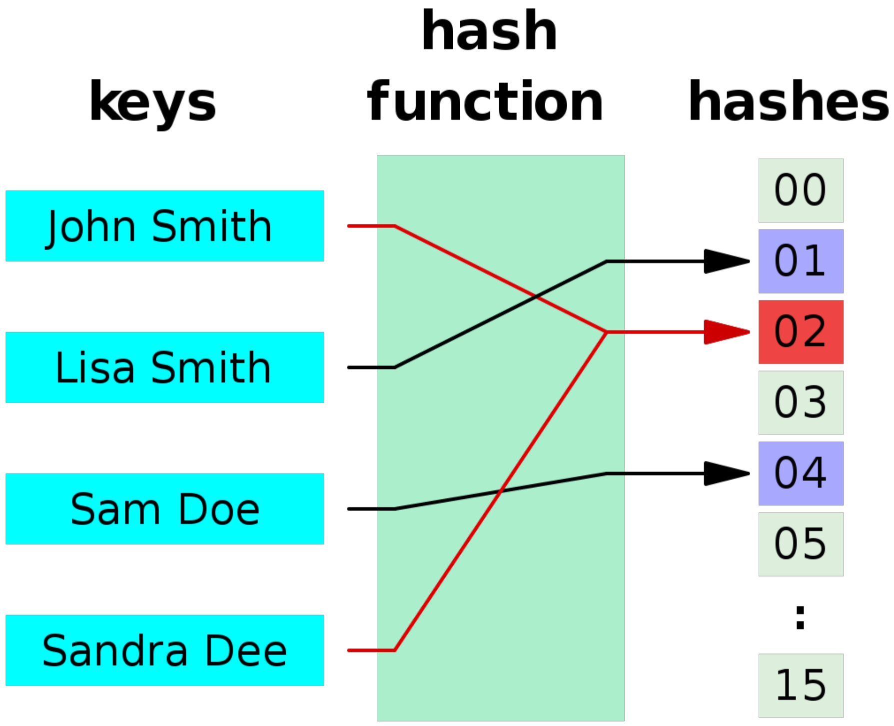
Feature selection

- Bi and Tri-gram for dynamic and 256-gram for static features [2][3]:
  - Binary n-gram?
    - One of the most effective and practical method for sequential data analysis
      - such as natural language processing (nlp), signal or sound processing, etc
    - Build “n length” tokens and count them all
    - Example of 3-gram:
      - for the data as follows: [apple, banana, orange, pear, mango]
        - we can obtain.. [apple, banana, orange], [banana, orange, pear], [orange, pear, mango]
      - it can apply for char-unit: [app, ppl, ple, leb, eba, ban, ana, nan, ana, …, ngo]
      - Then, count that tokens
        - For the tokenized data [app, ppl, ple, leb, eba, ban, ana, nan, ana, …, ngo],
        - n-gram table below would be obtained

app	ppl	ple	leb	eba	ban	ana	nan	…	ngo
1	1	1	1	1	1	2	1	…	1

- Feature selection

- Dimension reduction with Feature Hashing from 256-gram:
  - Data refined using 256-gram has more than 50,000 dimension..
  - Therefore, we apply Feature Hashing to that high dimensional vector
    - and obtained 1,000 ~ 10,000 dimensional vector



- Feature selection

- TFIDF of strings (with readability checker) [4]
  - Using printable characters between ascii code (33~125)

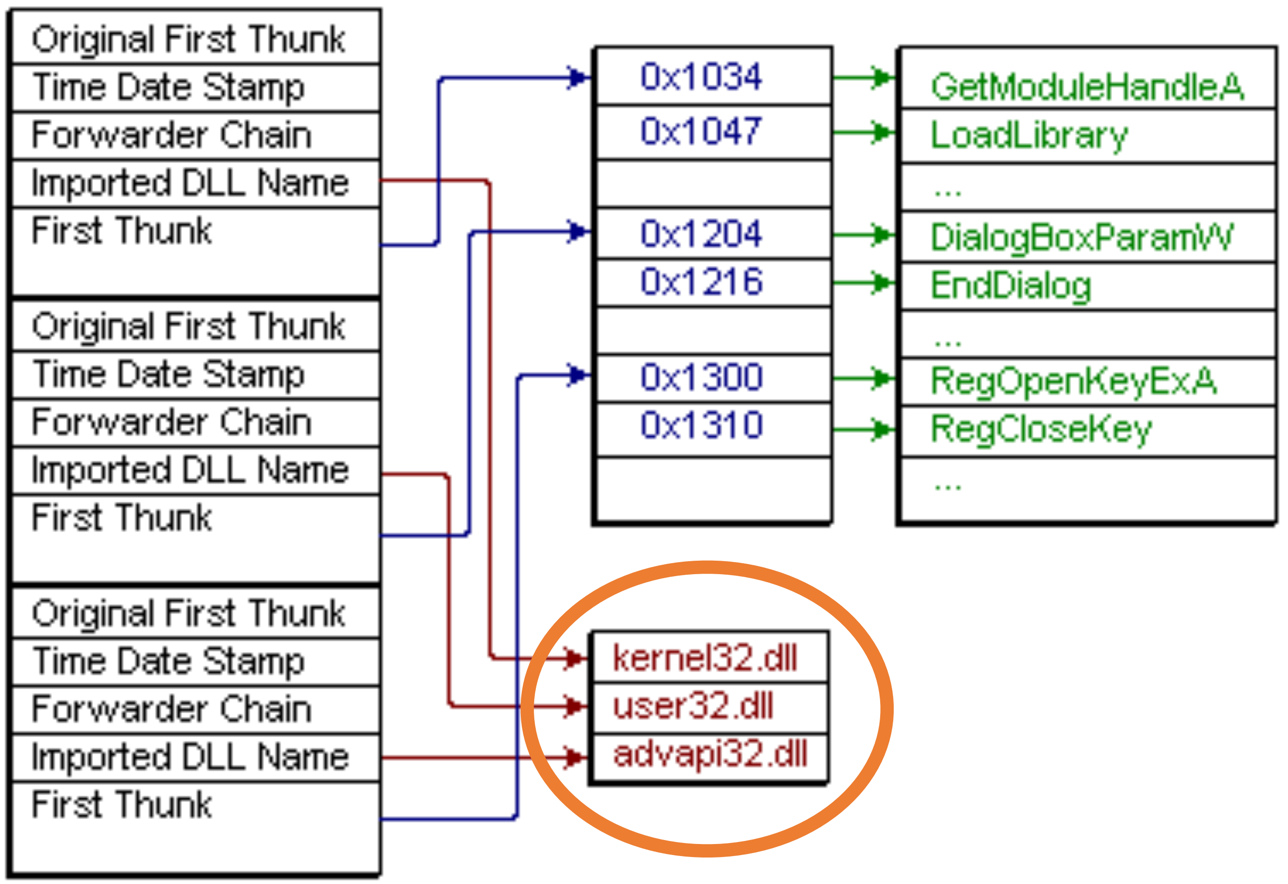
  - Readability checker?
    - Originally, it was applied to detection of malicious javascript files
    - Definition of readable words:

If it is > 70% alphabetical, has 20% < vowels < 60%, is less than 15 characters long, and does not contain > 2 repetitions of the same character in a row.

    - ex)
      - Respectfulness (O)
      - Dictionary (O)
      - sdifad13202 (X)

- Feature selection

- TFIDF of imported DLL (using pefile)



- Feature selection

- TFIDF of imported DLL :
  - TFIDF? Term Frequency - Inverse Document Frequency
    - is a numerical statistic intended to reflect how important a word is to a document in a collection or corpus
    - That is, this method originally invented for text analysis
    - For it is very useful for many types of data, we also applied it for malware detection

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

N = total number of documents

- Feature selection

- Term Frequency - Inverse Document Frequency
  - example)
    - Given 3 sentences, (from “https://nesoy.github.io/articles/2017-11/tf-idf”)
      - I love dogs.
      - I hate dogs and knitting.
      - Knitting is my hobby and my passion.
    - make a frequency table as below

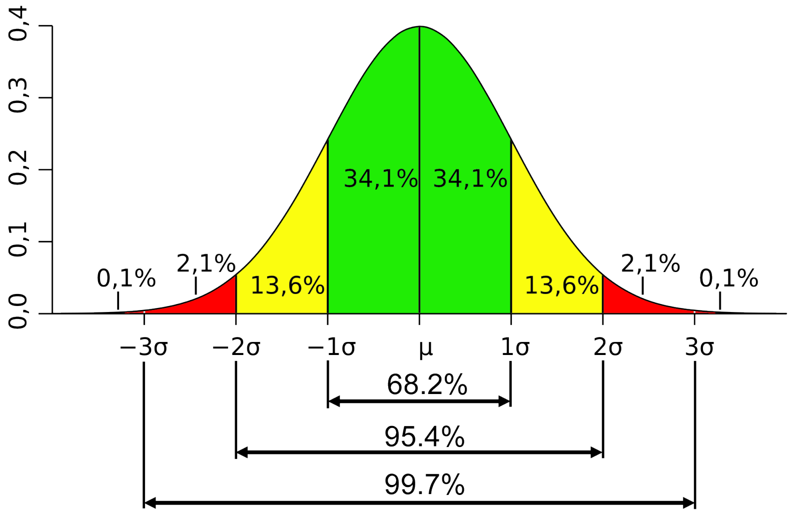
	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	1	1	1							
Doc 2	1		1	1	1	1				
Doc 3					1	1	1	2	1	1

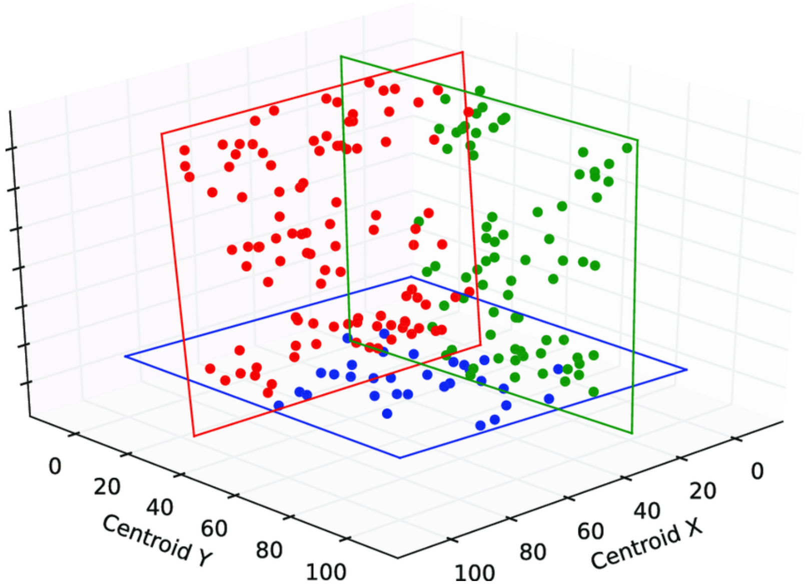
    - Then, calculate the importance of each word

	I	love	dogs	hate	and	knitting	is	my	hobby	passion
Doc 1	0.18	**0.48**	0.18							
Doc 2	0.18		0.18	**0.48**	0.18	0.18				
Doc 3					0.18	0.18	**0.48**	**0.95**	**0.48**	**0.48**

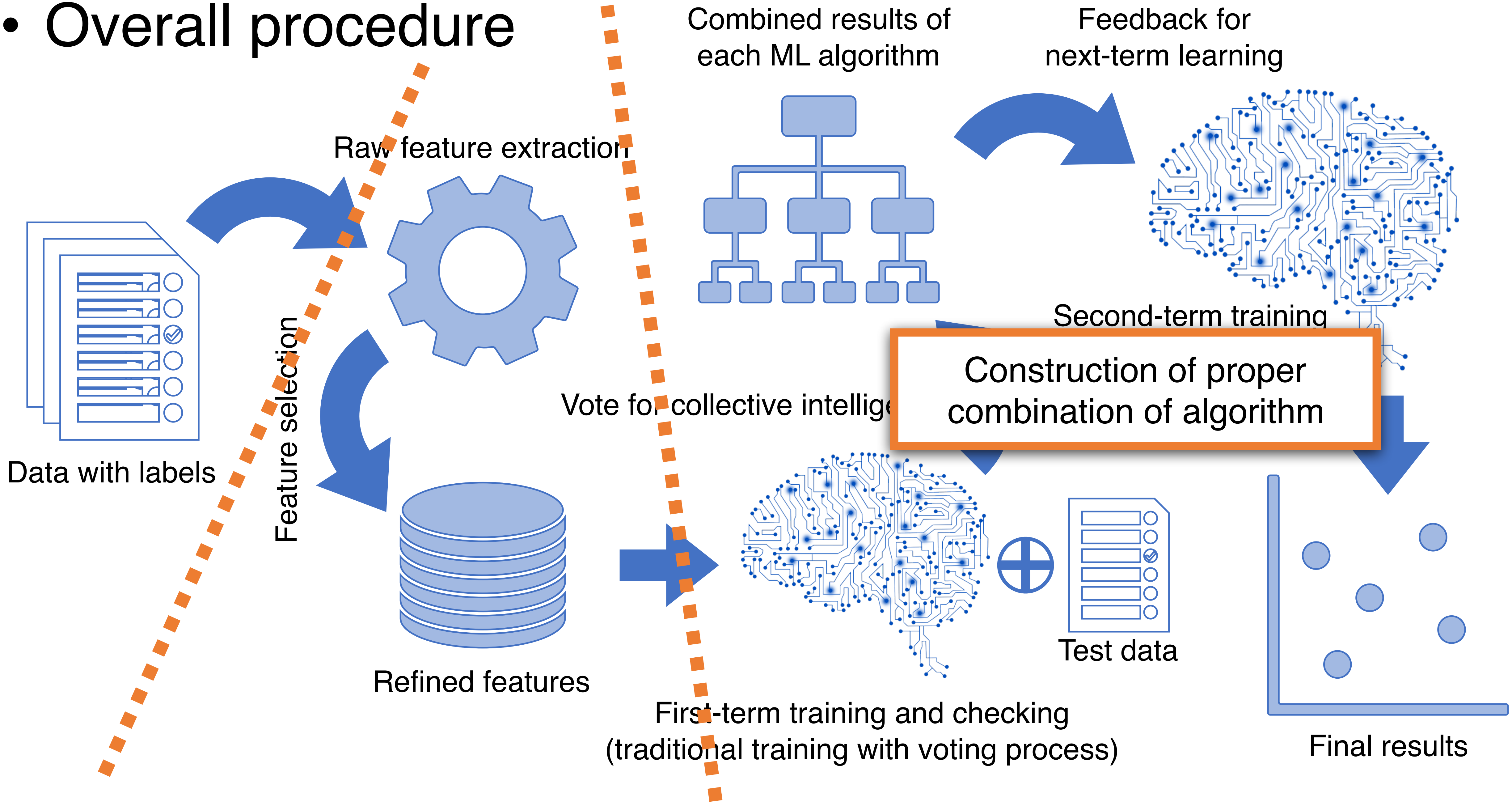
- Feature selection

- Feature selection methods
  - Removing features with low variance
  - Feature selection using “SelectFromModel”
    - (in scikit-learn)
    - We used “ExtraTreesClassifier()”
    - This method uses training algorithm itself to measure importance of features
    - Then, we could obtain 30~90 important features
  - Heuristically..
    - For this challenge,
      we excluded time-consuming features

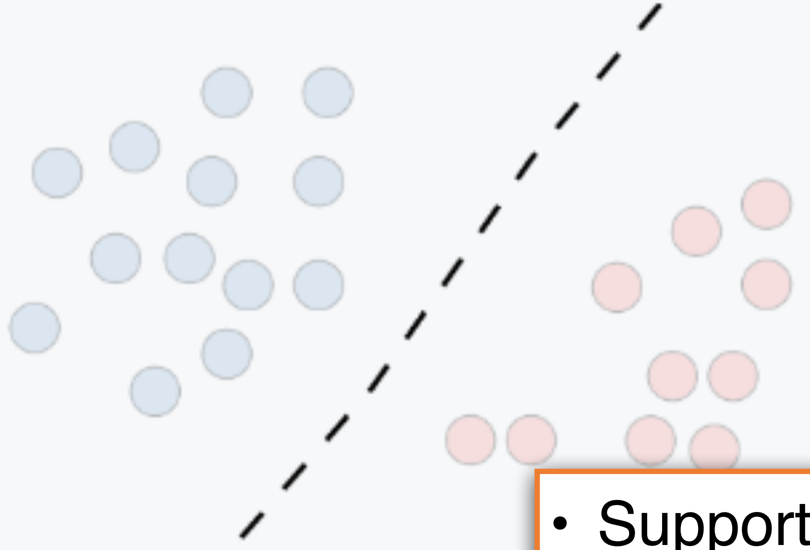
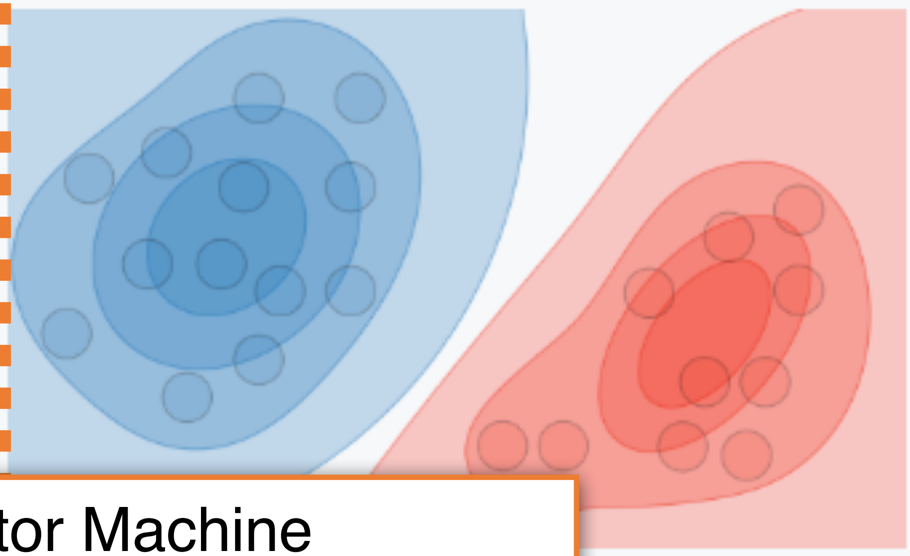




- Overall procedure

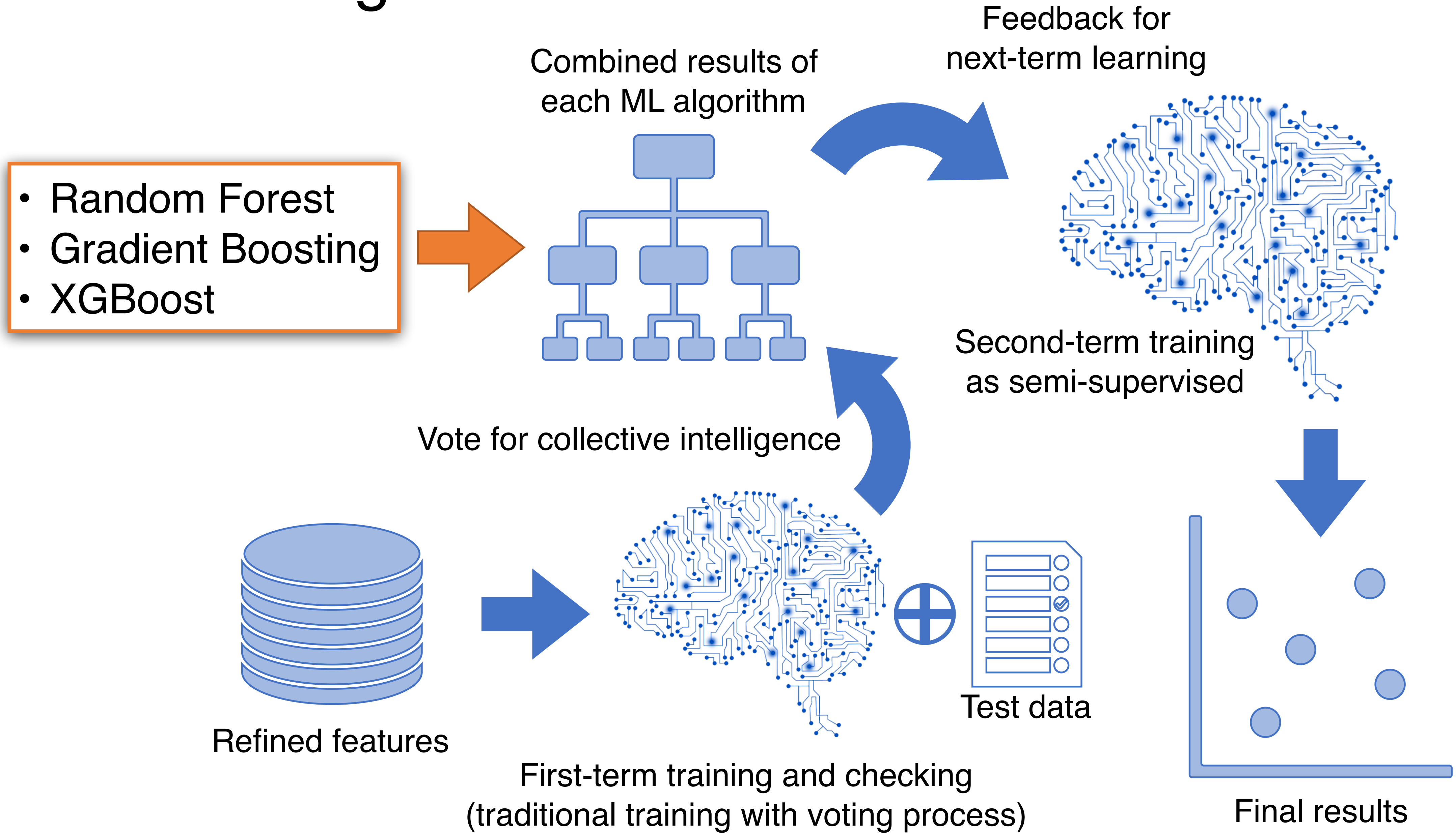


- Detection algorithms

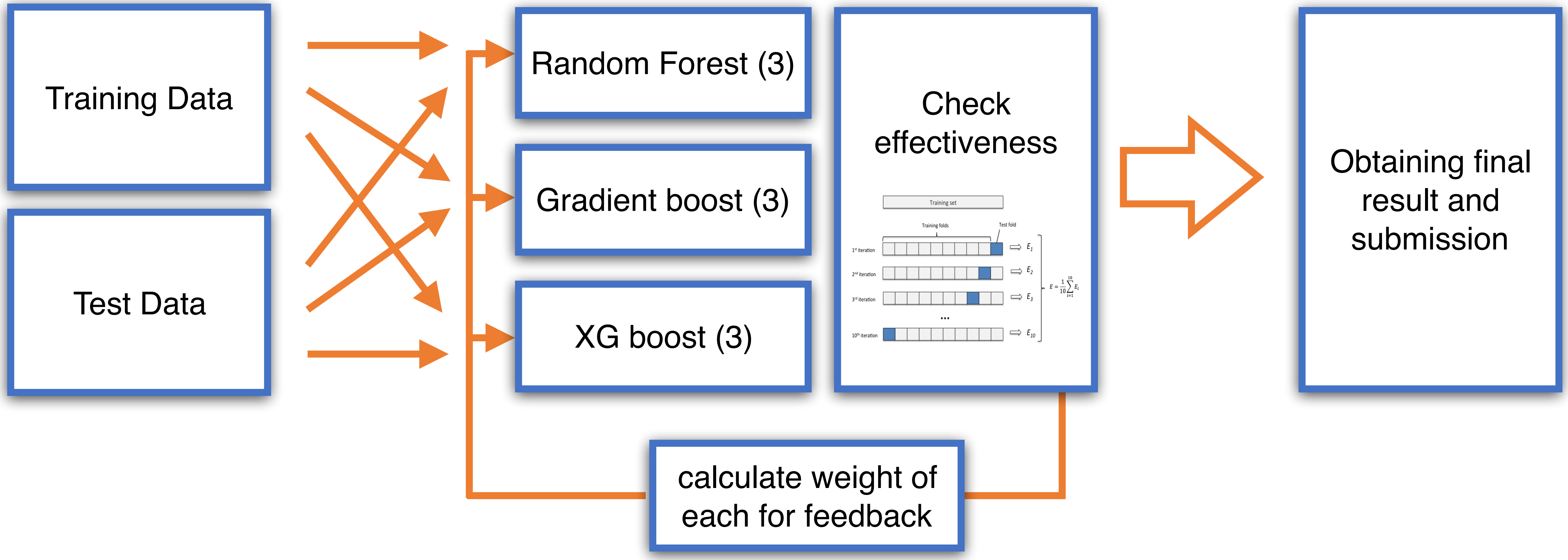
	Discriminative model	Generative model
Goal	Directly estimate $P(y\|x)$	Estimate $P(x\|y)$ to then deduce $P(y\|x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		
Examples	Regressions, SVMs	

- Support Vector Machine
- Random Forest (Decision Tree)
- Gradient Boosting (Ada Boost)
- XGBoost

Detection algorithms



Detection algorithms



$$Wn = \frac{R_n}{R_1 + R_2 + R_3}$$

Detection algorithms

$$Wn = \frac{R_n}{R_1 + R_2 + R_3}$$

Such that, W1 + W2 + W3 = 1

If results of RF, GB, XGB are 93, 95, 97 respectively, weights of RF is 93 / (93+95+97)

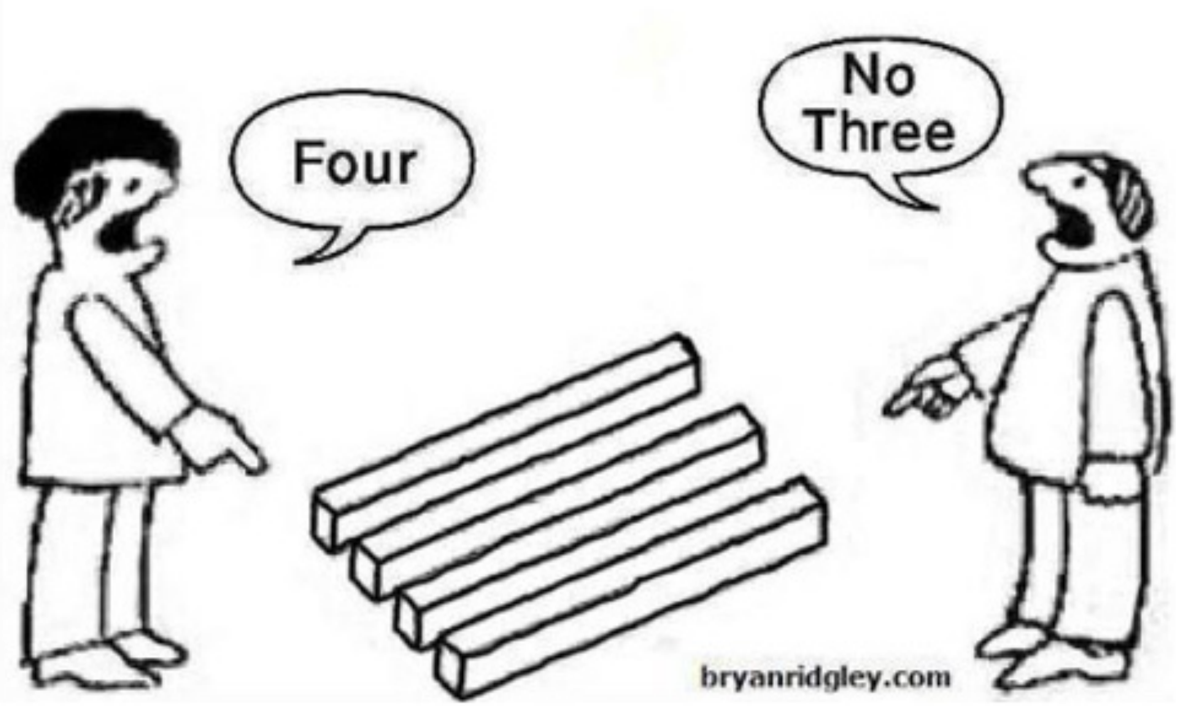
```python
def overall_Res(self, mRes, mWeight):
        sum_RF = np.mean(mRes['RandomForest'], axis=0)
        sum_XGB = np.mean(mRes['XGB'], axis=0)
        sum_GB = np.mean(mRes['GradientBoosting'], axis=0)

        RF = np.multiply(sum_RF, mWeight['RandomForest'])
        XGB = np.multiply(sum_XGB, mWeight['XGB'])
        GB = np.multiply(sum_GB, mWeight['GradientBoosting'])

        overall = np.sum([RF, XGB, GB], axis=0)

        return np.transpose(overall)
```

Give weight of 0.4



Give weight of 0.6

• References:

- [1] Kumagai, Atsutoshi, and Tomoharu Iwata. "Learning Dynamics of Decision Boundaries without Additional Labeled Data." Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, 2018.
- [2] Ahmadi, Mansour, et al. "Novel feature extraction, selection and fusion for effective malware family classification." Proceedings of the sixth ACM conference on data and application security and privacy. ACM, 2016.
- [3] Raff, Edward, et al. "Malware detection by eating a whole exe." arXiv preprint arXiv:1710.09435 (2017).
- [4] Readability: Likarish, Peter, Eunjin Jung, and Insoon Jo. "Obfuscated malicious javascript detection using classification techniques." *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*. IEEE, 2009.

- # Full-references:

[1] Sejnowski, Terrence J. "Higher−order Boltzmann machines." AIP Conference Proceedings. Vol. 151. No. 1. AIP, 1986.

[2] Hearst, Marti A., et al. "Support vector machines." IEEE Intelligent Systems and their applications 13.4 (1998): 18-28.

[3] Friedman, Jerome H. "Greedy function approximation: a gradient boosting machine." Annals of statistics (2001): 1189-1232.

[4] Rätsch, Gunnar, Takashi Onoda, and K-R. Müller. "Soft margins for AdaBoost." Machine learning 42.3 (2001): 287-320.

[5] Liaw, Andy, and Matthew Wiener. "Classification and regression by randomForest." R news 2.3 (2002): 18-22.

[6] John, George H., and Pat Langley. "Estimating continuous distributions in Bayesian classifiers." *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1995.

[7] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. ACM, 2016.

[8] Kumagai, Atsutoshi, and Tomoharu Iwata. "Learning Dynamics of Decision Boundaries without Additional Labeled Data." Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM, 2018.

[9] https://en.wikipedia.org/wiki/Power_law.

[10] Joachims, Thorsten. A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization. No. CMU-CS-96-118. Carnegie-mellon univ pittsburgh pa dept of computer science, 1996.

[11] Lee, Daniel D., and H. Sebastian Seung. "Algorithms for non-negative matrix factorization." Advances in neural information processing systems. 2001.

- # Full-references:

[12] Weinberger, Kilian, et al. "Feature hashing for large scale multitask learning." arXiv preprint arXiv:0902.2206 (2009).

[13] Likarish, Peter, Eunjin Jung, and Insoon Jo. "Obfuscated malicious javascript detection using classification techniques." Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on IEEE, 2009.

[14] Ahmadi, Mansour, et al. "Novel feature extraction, selection and fusion for effective malware family classification." Proceedings of the sixth ACM conference on data and application security and privacy. ACM, 2016.

[15] Karthikeyan, L., G. Jacob, and B. Manjunath. "Malware images: Visualization and automatic classification." Proceedings of the 8th International Symposium on Visualization for Cyber Security. 2011.

[16] Bradski, Gary, and Adrian Kaehler. "OpenCV." Dr. Dobb's journal of software tools 3 (2000).

[17] Raff, Edward, et al. "Malware detection by eating a whole exe." arXiv preprint arXiv:1710.09435 (2017).

- Thank you