Sketch Recognition with Few Examples

by

Kemal Tuğrul Yeşilbek

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Science and Engineering

July 04, 2016

Sketch Recognition with Few Examples

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Kemal Tuğrul Yeşilbek

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Assoc. Prof. T. Metin Sezgin, Koç University

_____

Assoc. Prof. Yücel Yemez, Koç University

_____

Asst. Prof. Ayşe Küçükyılmaz, Yeditepe University

Date: _____

To my mother, my dad, and my fiancée

ABSTRACT

Sketch recognition is the task of converting hand-drawn digital ink into symbolic computer representations. Since the early days of sketch recognition, the bulk of the work in the domain has focused on building accurate recognition algorithms for specific domains, and well defined databases. These lines of work adopt traditional machine learning approaches. They assume the presence of a fixed set of symbol classes, and availability plenty of annotated examples per class. However, in practice, these assumptions do not hold. In reality, the designer of a sketch recognition system starts with no labeled data at all, and faces the burden of data annotation. In this work, we propose to alleviate the burden of annotation by building systems that can learn from very few labeled examples, and large amounts of unlabeled data. Our systems perform self-learning by automatically extending a very small set of labeled examples with new examples extracted from unlabeled sketches. The end result is a sufficiently large set of labeled training data, which can subsequently be used to train classifiers. We present four self-learning methods with varying levels of implementation difficulty and runtime complexities. One of these methods leverages contextual co-occurrence patterns to build verifiably more diverse set of training instances. Rigorous experiments with large sets of data demonstrate that this novel approach based on exploiting contextual information leads to significant leaps in recognition performance.

ÖZETÇE

Çizim tanımanın ilgilendiği alan el ile çizilen dijital mürekkepleri sembolik bilgisayar ifadelerine çevirmektir. Çizim tanımanın ilk günlerinden beri alandaki çalışmaların çok büyük bir bölümü belirli ve iyi tanımlanmış veri tabanları için isabetli tanıyıcı modellerini geliştirmek üzerine odaklanmıştır. Bu gibi çalışmalar geleneksel makine öğrenmesi yaklaşımlarını kullanmaktadır. Bu yaklaşımlarda, önceden belirlenmiş sınıfların ve her sınıf için çokça etiketli örneğin bulunduğu varsayılır. Fakat pratikte bu varsayımların karşılanması güçtür. Gerçekte bir çizim tanıma sistemini geliştiren tasarımcının elinde hiç bir etiketli veri yoktur ve veri etiketleme külfeti ile karşı karşıyadır. Bu çalışmada etiketleme külfetini hafifletmek amacı ile geliştirilen, az sayıda etiketli ve çok sayıda etiketsiz veri kullanarak eğitilen sistemler sunmaktayız. Sunduğumuz sistemler çok küçük boyuttaki etiketlenmiş örnek kümesini öz-öğrenme yöntemleri ile etiketsiz örnekler arasından yeni örnekleri etiketleyerek otomatik olarak genişletmeyi hedefler. Otonom etiketlemenin sonucunda yeterli boyutta etiketli eğitim verisi ortaya çıkar ve bu veriler ile sınıflandırıcılar eğitilebilir. Bu çalışmada uygulaması ve çalışma zamanı karmaşıklığı birbirinden farklı dört öz-öğrenme yöntemi sunulmuştur. Sunulan yöntemlerden biri içeriksel eşdizimli örüntüleri kullanarak, tasdik edilebilir düzeyde daha çeşitli örnekler içeren eğitim örnek kümeleri elde etmektedir. Büyük veri setleri ile yürütülen titiz deneyler sonucu, sunulan içeriksel tabanlı özgün yöntemin tanıma performansını önemli ölçüde yükselttiği gözlemlenmiş ve raporlanmıştır.

ACKNOWLEDGMENTS

I would like to thank to people who supported me during my study.

Firstly and foremost, I would like to thank my advisor T. Metin Sezgin the time he spent for guiding me. Thank you for insightful feedbacks, guidance, and everything you taught me.

I would like to thank my thesis committee members, Yücel Yemez and Ayşe Küçükyılmaz for their valuable input.

I am most grateful to my parents Asuman Yeşilbek and Tuğrul Yeşilbek for their constant support through my education life. I would not be able to study this far and reach my goals without them.

I would also like to express my deepest gratitude to my fiancée Elif Özgen. Thank you for always being there for me. I very much appreciate your support.

I would also like to thank Şerike Çakmak, Neşe Alyüz, Cansu Şen, Bekir Berker Türker, Cağla Cıg, Ozan Can Altıok, Kurmanbek Kaiyrbekov, and all the colleagues at my research group.

Last but not least, I would like to thank all the students who participated in data collection and processing phases of this study.

This study is conducted for the project funded by TUBITAK under number 113E059.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

SVM	Support Vector Machine
N.N.	Nearest Neighbor
I.W.	Instance-Wise
M.o.D./MoD	Mean of Distances
AIG	Artificial Instance Generation
#+	Number of Annotated Sketch
α	Context Weight
IDM	Image Deformation Model
CPU	Central Processing Unit

Chapter 1

INTRODUCTION

1.1 Motivation

Sketching is a powerful medium of communication. Some ideas can be expressed using sketches clearer and easier than any other medium of communication. Through the history, humans expressed their ideas by sketching. This essential way of communication was studied by many researches over the years and found many applications in the field of sketch recognition. From the arise of sketch recognition to present, researches collected different data sets, developed new applications, and brought different approaches to the field. Most of the researches have focused on exploring methods that work better for predetermined classes within offline settings using lots of data. Even tough it is now cheaper to collect and process large amount of data, the big data is not always available. In fact, there exist important problems where the available data is scare.

We believe that when sketch recognizers can be trained using few examples and yield sufficient performance, it will be a precious tool for applications that utilize sketch recognition. An example to applications that benefit from sketch recognition trained with few examples, can be sketch recognizers which can extend its set of supported classes with minor user interaction. For such recognizers, it will be possible to add new class support via a quick interaction of the user. Even though large data sets with lots of classes, like the data set collected by Eitz et. al. [[Eitz et al., 2012]], can be collected, there will never be enough classes for every possible sketch object. It is because people may have their own unique marks and drawings to describe

particular objects or actions. In this context the users' ability to add new class support to have personal sketch recognizers, thus sketch recognition with few examples is an important area to study.

In this work we restrict ourselves around a few settings. Firstly, we focus on developing recognizers to work on scene sketches instead of isolated sketch objects (i.e. circuit diagram versus a single resistor). Scene sketches may contain multiple sketch objects (i.e. circuit components) or sketch pieces that are not meaningful. Sketch recognition in scene sketches are considered as a more challenging problem. In this document, we use the expressions of scene sketches and sketches interchangeably. Secondly, we investigate approaches that can operate when only few (1-3) sketches are annotated. Lastly, we assume that the system has access to unannotated sketches that are similar to the annotated ones. The settings we mentioned yield semi-supervised setting, thus we build our work around semi-supervised learning.

We investigated and proposed multiple methods for sketch recognition with few examples problem. Our methods are mainly based on self-training and transductive learning. Self-training methods we investigated aims to label some of the unlabeled instances prior training a final classifier. Transductive learning approach we employ do not separate the data set as training and testing sub-sets but let all of the instances be visible to the learners but not with their labels. In order to put our methods to test, we created a pipeline to mimic and analyze the end-to-end working of our methods. Furthermore, we collected a scene sketch data set to use in our experiments. The collected data set contains 1,522 sketches.

While comparing methods, we set nearest neighbor without self-training, as our baseline. Although nearest neighbor is a very simple method, it achieves better performances than many other methods. Whereas nearest neighbor achieves good performances, our novel context based self-training method achieves better recognition rates. This novel method utilizes the placement patterns of sketch objects in sketches along with their appearances. The methods we proposed are not specialized for certain sketch types or data sets, and can operate with sketches with different structures

and properties.

In the next chapter, we introduce our data set and discuss its properties. In the third chapter, we explain the pipeline we developed in detail and propose approaches that are employed to attack the few example problem. In the fourth chapter, we explain and report the self-training methods we utilize in our study. In the last chapter, we present the overall results of our study and discuss our findings.

1.2 Problem Statement and Approach

We define our work formally as scene sketch object recognition via self-training where few sketches are annotated within many unannotated sketches. An example scenario fits to this definition is as follows: there are multiple unannotated sketches of the same kind (i.e. circuit diagrams) and user wants to create a sketch recognizer for a particular sketch object (i.e. battery component). The recognizer to be created should be able to distinguish between the designated sketch objects and other sketch pieces. In order to create such recognizer, user annotates sketch objects that are examples of the desired class in few sketches. The user's annotation is insignificant in terms of labor as annotating a single object in a sketch takes only few seconds.

We do not enforce any restrictions on the number of sketches to be annotated. However, in this work we analyze settings where one, two, or three sketches are annotated, which is a challenging task. With the annotation, the designated sketch object instances will be labeled as positive and other sketch pieces in the same sketch will be labeled as negative. In this document, we refer the object class that is designated to be detected in sketches as positive class, and the other sketch pieces as negative. The remaining sketch pieces that are members of unannotated sketches will remain unlabeled.

As unlabeled instances are present, our problem poses a semi-supervised setting. Thus, we focus on semi-supervised learning in our work. Furthermore, dividing instances as positive and negative emerge binary classification. Our approach do not utilize any transferred information or pre-learned models. We practice transductive

learning, which is a common approach in semi-supervised learning. In transductive learning, the data set is not separated as training or testing but all instances are visible to the learner. However, while instances are visible, only some of the instances' labels are available for the learner.

There are multiple ways to approach this problem. We build our work around semi-supervised learning as it is shown that utilizing unlabeled instances may improve the classification performances. Among the semi-supervised methods, we concentrate on self-training as it is cost-effective and does not make assumptions on data set apart from smoothness. While working with few labeled instances, we believe that complicated assumptions will not apply well as there is inadequate room for generalization.

1.3 Related Work

We study minimal label sketch recognition, based on semi-supervised machine learning methods. Therefore, we will discuss related work on sketch recognition, and machine learning.

1.3.1 Machine Learning

Although sketch recognition community never leaned on learning with few examples, machine learning community studied this topic in supervised learning setting. The studies for learning with few examples starts when the training data was scare, and the computational machines were not powerful to process large amount of data.

One of the earliest work is done by Jain et. al. in 1982 [[Jain and Chandrasekaran, 1982]]. In their chapter, they discuss the effects of dimensionality and the small example set. Similar to Jain et.al., Raudys et. al. sketched out the problems with small sample size and proposed directions to take for working on small sample sizes [[Raudys and Jain, 1991]].

In their work, Skurichina et. al. studied classifiers on data sets that have less instances than the number of features [[Skurichina and Duin, 1996]]. They reported

that bagging yields improved classifiers. In our work we also found that bagging improve classifier performance.

In their work, Wolf et. al. introduced a method for increasing the number of instances via synthetically generated instances in feature space along with fast bootstrapping [[Wolf and Martin, 2005]]. A similar work to their was proposed by Chawla et. al. in 2002 which aimed to reduce the imbalance in data set by synthetically generating new examples in feature space [[Chawla et al., 2002]].

In 2006 and 2007, Fei et. al. demonstrated that it is sufficient to have few examples to train an image classifier by transferring information from other object classes [[Fei-Fei et al., 2006]] [[Fei-Fei et al., 2007]].

In 2007, Zhou et. al. proposed a multiple view approach for semi-supervised learning with few labeled examples [[Zhou et al., 2007]]. In their work, multiple different feature sets was used to further improve the performance of semi-supervised learning method. While multiple view approaches are popular in semi-supervised learning, our setting was impotent to utilize such methods.

In 2011, Salperwyck et. al. analyzed and reported the performances of common classifiers on different data sets [[Salperwyck and Lemaire, 2011]].

1.3.2 Sketch Recognition

While there are numerous work in machine learning for learning with few examples, we were unable to find any work in the area of sketch recognition. The closest work we found is by Miller et. al. where they were able to generate recognizers using few examples [[Miller et al., 2000]]. In their work, they learned and transferred parameter range for linear geometric transformations and used them to generate synthetic instances.

There are few work considering semi-supervised learning for sketch recognition. One of the work is by Yanik et. al.. In their work, they explore active learning for sketch recognition [[Yanık and Sezgin, 2015]]. Another work is by Tirkaz et. al. In their work, they propose a method for auto-completion for sketch recognition

which allow recognition systems to predict the classes of the incomplete sketches [[Tirkaz et al., 2012]].

Chapter 2

DATA SET

The data used in machine learning studies affect both the performance and the credibility of classification methods. That is why collecting a realistic data set is important. Realistic data collection means collecting the data in an environment where the conditions are as close as possible to the conditions where the recognizers would perform in. In this context, collecting the data in users', or systems', natural habitat (in-wild) is important. In order to develop and measure the recognizer's performance without bias, realistically collected data is required.

We had four key practices in order to collect the data realistically. Firstly, instead of inviting participants to our office/laboratory, we visited them in their own environments. Secondly, we used commercially available tablets which have both active stylus support and sketch on the screen capability to supply a proper sketching environment. Thirdly, instead of asking participants to copy a sketch, which would lead to identical sketches, we asked them to solve given questions. With asking for solutions to given questions, we assured that participants use the same sketch objects in their solutions even if their solutions were different. Lastly, we asked from participants to solve a question only once to prevent duplicate sketches in the data set.

To collect data, we visited 12 schools range from primary school to university. We collected 1,522 answers from 1,034 students. We directed different questions to students in different education levels. For instance, while primary school students answered logic problems, junior computer engineering students answered computer science questions. For each education level, we studied associated syllabus while preparing the questions. We prepared seven questions in total. The list of questions and their associated targeted education levels are presented in Table 2.1. We tried

Question	Sketch Count
Balance	463
Money	379
Reflection	289
Par. Circuit	112
Ser. Circuit	110
Tree	47
Box-Pointer	122

Table 2.2: Sketch Count Distribution

to prepare questions that would lead to complex sketch objects in their solutions but found it difficult to find such questions for primary school students. In Table 2.2, we present the number of sketches collected for each question.

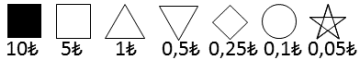
Question	Education Level
Draw a scale in balance using the following objects. You cannot have duplicate objects in your sketch. 	Primary School
Draw the change amount using the following objects for 10₺ given for a 6,55₺ item. 	Primary School
Draw the reflections of parallelogram on x axis, trapezoid on y axis, and square on origin. 	Primary School
Draw a circuit connected in parallel that has three resistors and one battery.	Mid/High School
Draw a circuit connected in series that has three batteries, two lamps, and one switch.	Mid/High School
Draw the abstract syntax tree of the given expression: ((1-6*3)/5)+2	Computer Science/Engineering
Draw the box-pointer diagram of the given list: '(1.(2.3))	Computer Science/Engineering

Table 2.1: Questions and Corresponding Education Levels

Question	**Classes**
Balance	Circle, Triangle, Square, Diamond, Star Bullet
Box-Pointer	Number, Arrow Right, Arrow Down, Double Box
Money	Circle, Triangle, Square, Star, Upside Down Triangle, Square, Plus
Par. Circuit	Resistor, Battery
Reflection	Parallelogram Right, Square, Parallelogram Left, Diamond, Trapezoid Up, Cross, Trapezoid Down
Ser. Circuit	Resistor, Battery
Tree	Number, Circle, Square, Plus, Minus

Table 2.3: Question-Class Pairs

The sketch objects we identified in our study per each question are presented in Table 2.3. Example drawings for the classes are represented in Table 2.4.

In order to measure the performance of the methods, we annotated each sketch to prepare ground truth. In order to annotate sketches, we developed a sketch annotator where user can load, fragment and annotate the objects in sketches. In order to supply fast and error-free annotations, we used gesture based selection for sketch objects and presented a list of possible classes. A screen-shot from the annotator is presented in Figure 2.1. During the preparation of ground truth, we did not restrict ourselves with the classes determined for the questions but tried to annotate sketches for objects that may be considered as classes in different studies. Example sketches from the data set are presented in Figure 2.2. Each stroke is painted with different color in the figure.

Class	Drawing	Class	Drawing
Circle		Upside Down Triangle	
Triangle		Plus	
Square		Resistor	
Diamond		Battery	
Star Bullet		Parallelogram Right	
Number		Parallelogram Left	
Arrow Right		Trapezoid Up	
Arrow Down		Trapezoid Down	
Double Box		Cross	
Star		Minus	

Table 2.4: Example Drawings for Classes

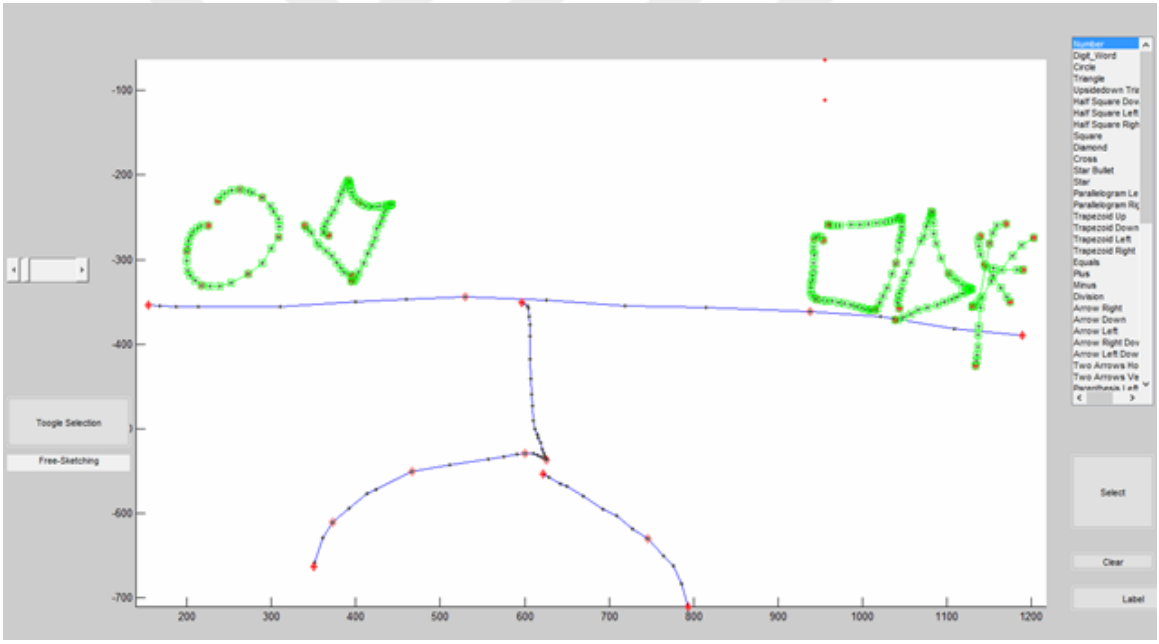


Figure 2.1: A Screenshot from the Annotator Software

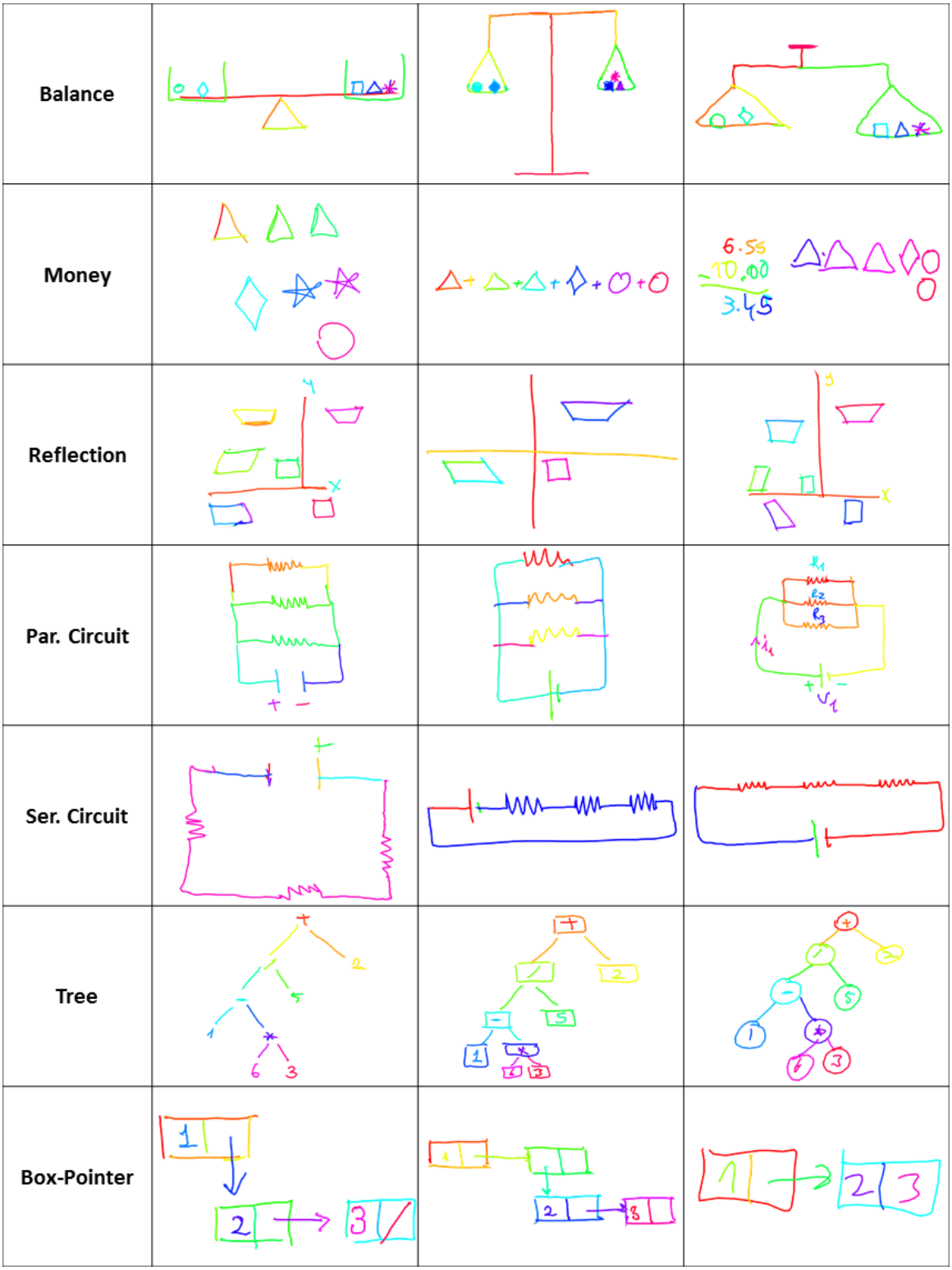


Figure 2.2: Example Sketches from Data set

Chapter 3

EXPERIMENT SETTING

We designed a set of experiments to analyze methods for sketch recognition with few examples. We analyze the methods with three independent, and one dependent variable. The independent variable is performance of a method. Performance indicates how successful a method performs for classification tasks. One of the dependent variables is the number of sketches annotated. This variable ranges from one to three. We expect the performances of methods to increase as the number of annotated sketches increase. There are different sketch objects present for different questions. Hence we examine each question separately, and this introduce another dependent variable to our experiments which is the question being examined. The sketches to be annotated are selected randomly. Because of this randomization, we repeat each question - annotation count pair for five times. When the values of dependent variables combine, it generates 105 experiments in total. The experiments are conducted in repeated measure. For our experiments, this means that each method is given the same data set with same annotated sketches. It is easier to compare different methods' performances as they all use the same data.

In order to perform our experiments we developed a pipeline. Using the pipeline, we are able to change the methods in any steps and analyze them easily. In this thesis, we explain our proposed solutions to different challenges and sketch our approach for this problem around this pipeline.

The pipeline we developed mimics the end to end working of our approach. In our approach, user annotates few sketches within unannotated sketches, and the system generate a recognizer as output. Our framework consist of four main steps. Those steps and their associated sub-steps can be decoupled and they can work indepen-

dently from the pipeline. The main steps are listed in order as follows:

1. Data Preparation
2. Conservative Rejection
3. Learning
4. Analyzation

The data preparation step processes the raw sketch data and transform it into a more structured form that learning tasks can be operated with. Conservative rejection step uses a simple but fast recognizer to pre-classify some of the instances as negative to have a smaller data set for fasten the learning processes in later steps. Learner step firstly labels some of the unlabeled instances as positive using self-training methods and trains a classifier. Lastly, the analyzation step compares the predicted labels with ground truth labels and reports performance results. Component diagram of the system is presented in Figure 3.1. All of the steps, except the analyzation, are parallelized in such way that each experiment can be submitted to a different CPU core using Sun Grid Engine [[Gentzsch, 2001]]. Each step is explained in detail within their associated sections along with the methods used.

3.1 Data Preparation

We preprocess sketches in four steps. In the first step, we fragment sketches into their primitives (arcs and lines). In the second step, we combine consecutive primitives to generate sketch pieces. Each generated sketch piece is considered as a single instance. In this document we use sketch piece and instance expressions interchangeably. In the third part, we extract features of the instances. In final step, we label instances to mimic the user annotation.

In scene sketches, multiple sketch objects may share the same stroke or single stroke may embody non-sketch pieces along with sketch objects. For example, users

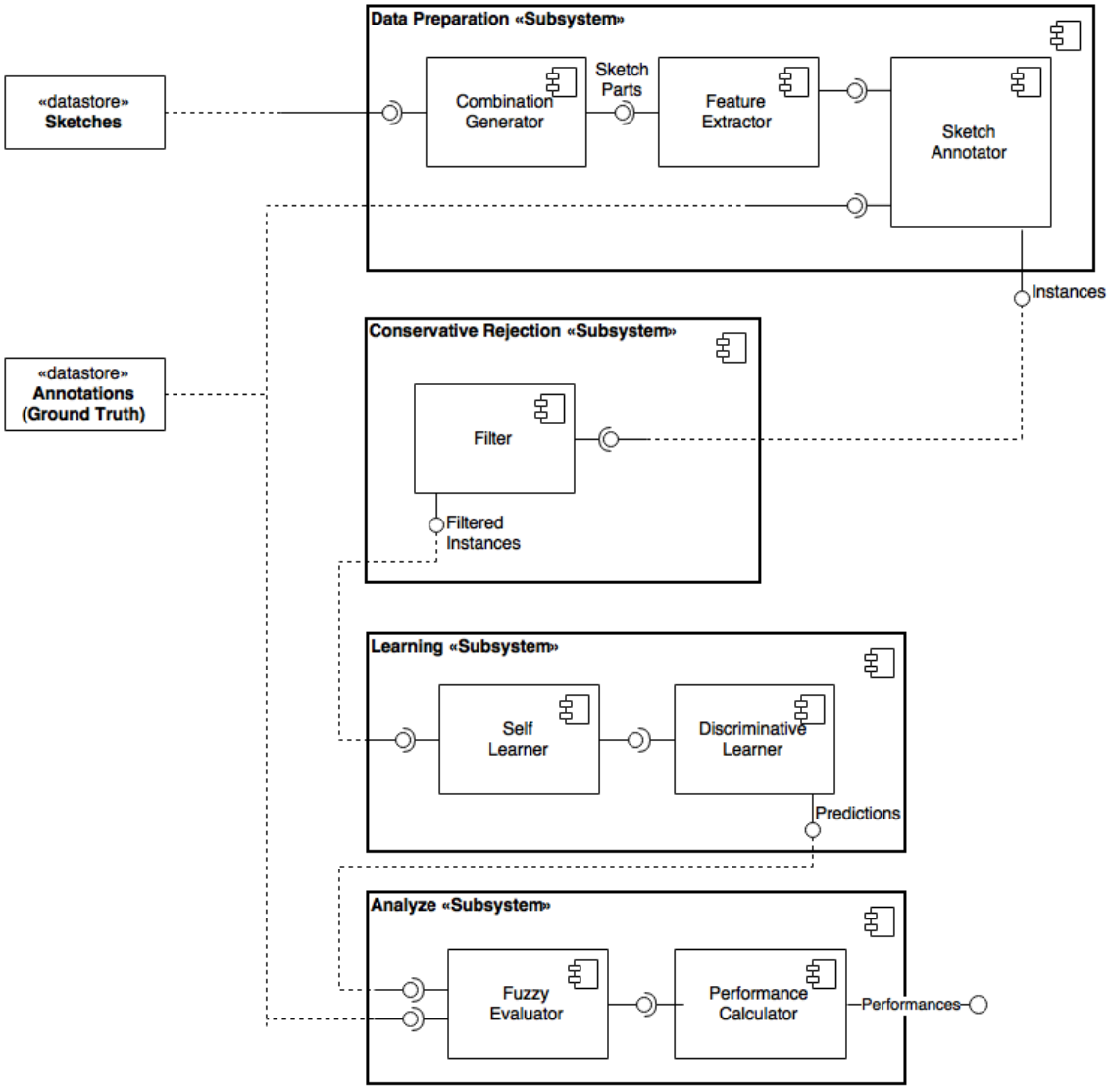


Figure 3.1: Component Diagram of the Framework

may draw an entire circuit diagram in serial circuit question with single stroke. In order to dissect the sketch, we fragment strokes using Douglas-Peucker algorithm [[Douglas and Peucker, 1973]]. With fragmentation, we break down strokes into primitive pieces: lines and arcs. The free-parameter required for Douglas-Peucker algorithm is set for each sketch independently during the preparation of ground truth by annotators.

Like multiple sketch objects can share the same stroke, multiple strokes can form a single sketch object. For example, a square can be drawn with a single or four strokes. In order to handle this condition, we combine consecutive sketch primitives. In this study, we combined two to 15 sketch primitives to form sketch pieces. After combination, each sketch piece is considered as an instance. We illustrate the fragment and combine process in Figure 3.2. The first phase in the figure shows the sketch with the starting point as green dot, and the sketching direction with arrows. In the second phase of the figure, the fragmentation points (or corners) are marked with red dots. The lines between each red dot are primitives. In third and fourth phases, the combinations generated by combination of consecutive primitives are presented. As can be seen, this single triangle generates four instances.

In our experiments, we use IDM features [[Ouyang and Davis, 2009]]. IDM feature set is a popular selection for sketch recognition and its effectiveness is shown in many studies. We borrow the best values for free parameters of IDM from the study of Tumen et. al. [[Tumen et al., 2010]]. The parameters we set yield a feature space that has 720 dimensions. We avoided to handcraft new features as we aim to propose a generic solution to sketch recognition with few examples. Thus we used this generic feature set instead of using data set specific features.

In the last step of data preparation, we mimic the user annotation. To perform that, we select a sketch in random, label the sketch objects as positive, label the sketch pieces that are not sketch objects as negative, and leave the sketch pieces that are members of other sketches as unlabeled. In condition where a sketch embodies multiple sketch objects, we randomly select one of them and label it as positive.

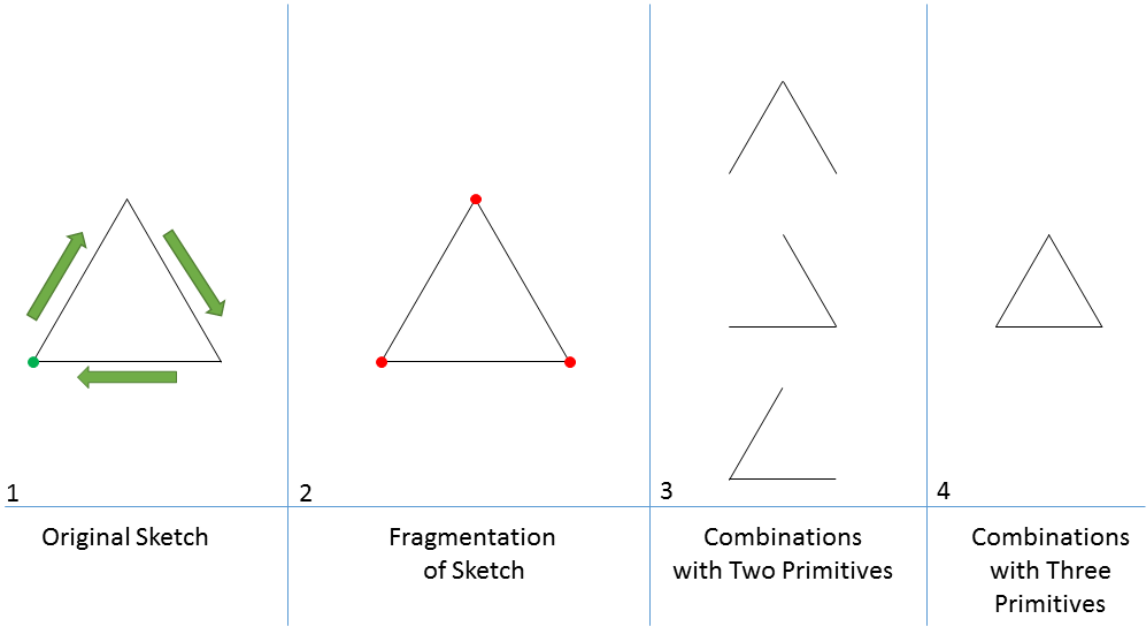


Figure 3.2: An Example for Sketch Fragmentation and Combination Process

The remaining sketch objects are marked as unlabeled. With this intervention, we eliminate the chance of having more than one positive labeled instance for each sketch. As we practice binary classification, we annotate sketch objects that are members of the positive class.

3.2 Conservative Rejection

Data preprocessing phase generates numerous of instances. With fragmentation and combination, each question yields tens of thousands of instances. This arises scalability concern for learning methods. Although there are vast number of instances, a large portion of those can be filtered out easily. For instance, if we define square as our positive class, it should be easy to predict the scale drawing in the balance question as negative. In order to reduce the instance count, we use an instance filter inspired by the work of Viola et. al. [[Viola and Jones, 2001]].

In this step, we train a simple but fast classifier from the labeled instances and classify the unlabeled instances using this classifier. We filter out a portion of the

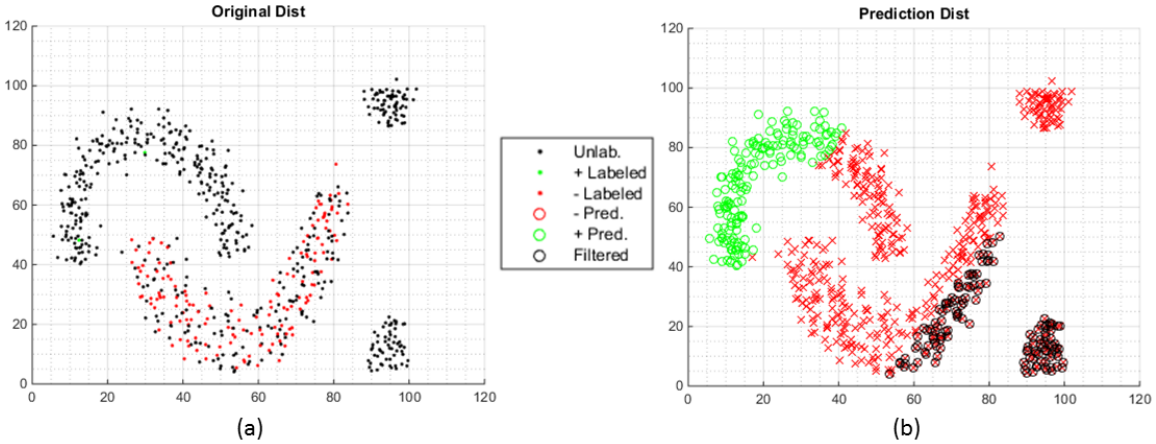


Figure 3.3: Conservative Rejection with Linear Kernel SVM in 2D Toy Data. The figure in (a) displays the input, and the figure in (b) displays the filtered out instances along with predictions.

negative predicted instances. The filtered out instances will not take place in later learning processes and will have negative prediction label in the evaluation step. The portion of the instances to be filtered out is determined by a free parameter. In our experiments, we set this parameter to 25%, meaning the quarter of the instances that are predicted as negative will be filtered out. The instances to be filtered out are selected among the instances that have the highest prediction scores.

In our experiments, we use linear kernel SVM for conservative rejection [[Cortes and Vapnik, 1995]]. We choose to use linear kernel due to its high speed and its superior generalization ability than other non-linear kernels when the labeled instance count is low. Although we choose to use this classifier, conservative rejection can be performed using any classifier that assign scores to predicted instances. An illustration of how conservative rejection with linear kernel SVM works on 2D toy data is presented in 3.3.

We report the performance of our conservative rejection method using false negative count and false omission rate in Figure 3.4 and 3.5 respectively. False omission rate is the ratio of false negative count over negative ground truth count. As can be seen in the results, even when we filter out a large portion of the instances, the falsely

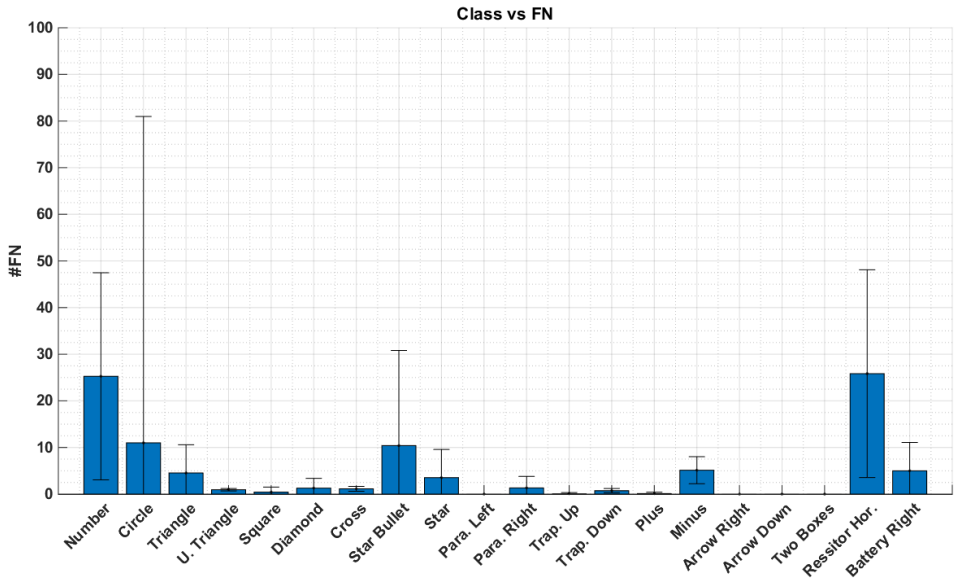


Figure 3.4: Performance of Conservative Rejection with False Negative Count

filtered instance count is very low.

3.3 Self-Training

Utilizing unlabeled data may help to build better recognizers [Zhu, 2005]. It is shown that when assumptions are met, self-training methods achieve better results than methods that use only labeled data. In our study, we use self-training methods in order to utilize the vast number of unlabeled instances in our data. Self-training is a commonly used method in semi-supervised learning studies. Basically, the self-training methods build a base classifier from the initial labeled data, and use this classifier to auto-label some of the unlabeled instances. We use self-training as it is cost-effective and does not make complex assumptions on the data. We believe that avoiding from complex assumptions and leaning on the smoothness assumption (i.e. close instances share the same class) is the best approach to take when it is desired to develop a generic recognizer scheme for few labeled instances.

While an annotated sketch yields single positive instance, it yields hundreds of

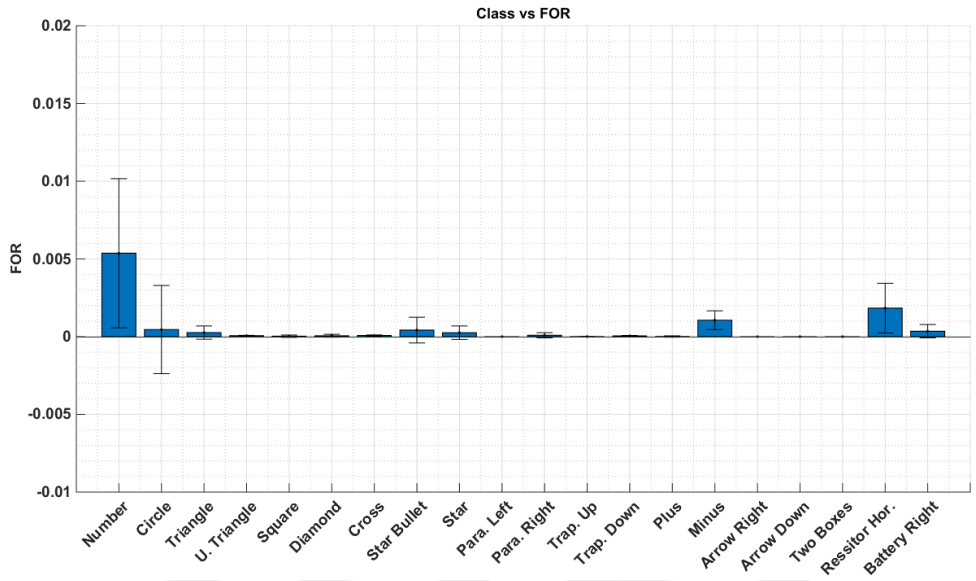


Figure 3.5: Performance of Conservative Rejection with False Omission Rate

negative instances. This brings high imbalance between positive and negative labeled instances. Because of this imbalance, we believe that self-training for negative class is not necessary. In addition to the large number of negative instances we have in the data set, there is the risk of introducing false negatives with self-training. Therefore we decided to perform self-training only for positive class. This means that we only increase the positive instance count with self-training.

One challenge self-training poses is to determine how many instances will be added to the labeled instance set. There are multiple approaches for this problem but none of them brings a solution [[Triguero et al., 2015]]. In our experiments we label only a subset of the unlabeled instances as labeling all of the data set will introduce numerous of false labeled instances. While it is reported that labeling subset approach works better than labeling all of the unlabeled instances, it adds a free parameter to decide: the number of instances to be labeled. It is a sensitive parameter as if it is high, the mislabeling chance will increase; if it is low, there will be less labeled instances. In our study, we tune this parameter for each experiment so that the labeled instance count

will be 15 at the end of self-training. This is rather a conservative value as there are much more than 15 sketches for each question. We explain and analyze self-training methods we used in detail in its corresponding chapter.

3.4 Discriminative Learning

We train the final binary classifiers in this step. While training, we both use the annotated labeled instances and the labeled instances the self-trainers yield. The classifiers trained in this step are used to make the final predictions on the instances. We adopt classical supervised classification methods for this step. We do not restrict ourselves with classifiers that are used for self-training as one classifier may be better for self-training while another may be better for classification.

We tested two different classifiers in our experiments: nearest neighbor and linear SVM with bagging. While nearest neighbor is a very simple classifier, it exhibits good performance in our setting. Linear SVM with bagging is a meta-learning method where it generates 50 random subsets of the data set and train models for each of those subsets. The subsets are generated by randomly selecting half of the instances from the data set.For prediction, we practice majority voting. Graphical explanation of the bagging method is presented in Figure 3.6.

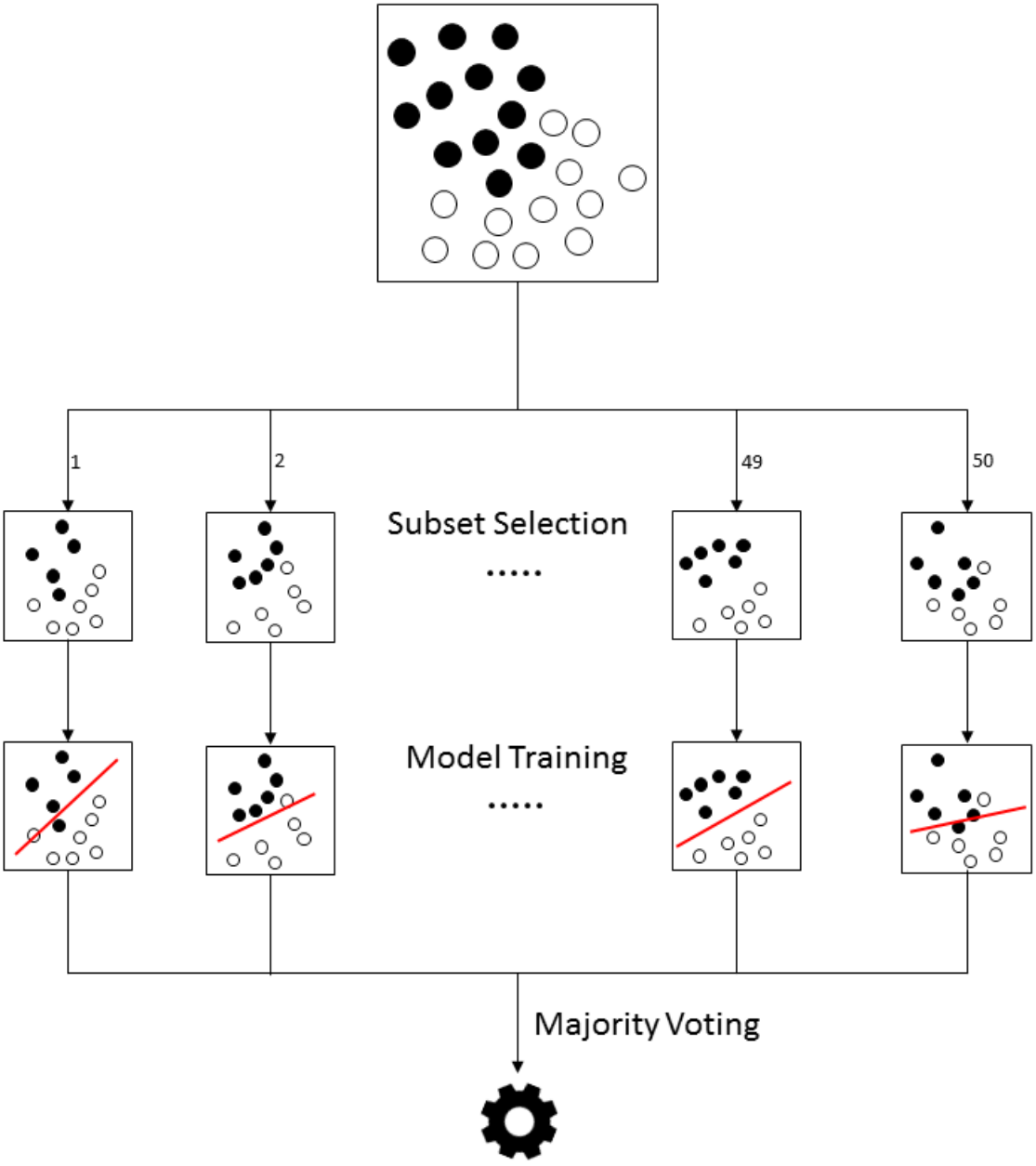


Figure 3.6: Graphical Explanation of Bagging

Chapter 4

SELF-TRAINING

We propose four approaches for self-training and increasing labeled instance set size. With increasing labeled set size, we aim to input more information to learners. In this chapter, we propose two nearest neighbor based self-training methods, a novel context based self-training method, and a method that increase the labeled instance set via introducing artificial instances. The nearest neighbor based methods select the unlabeled instances that are close to the labeled instances to label. Artificial instance generation method introduces instances that are synthetically generated based on the labeled instances. We explain each methods and present their performances in this chapter. Context based method utilize the placements of instances in sketches along with their appearance for self-training.

4.1 Instance Wise Nearest Neighbor

Instance wise nearest neighbor self-training approach aims to extend the positive instance set by labeling the unlabeled instances that are closest to each positive labeled instance. In order to perform this, each positive instance labels equal number of unlabeled instances that are closest to them. For example, if there are three positive instances and the positive labeled instance count is desired to be nine, each positive instance labels two unlabeled instances that are closest to them. This approach labels the instances that are the most similar to the positive labeled instances, which would result in rather lower false positive rate. Operation of this method is represented using 2D toy data in Figure 4.1. Note that the positive instance set size was two before self-training labeled 50 other instances.

In Figure 4.2, we present the performance of instance-wise nearest neighbor self-

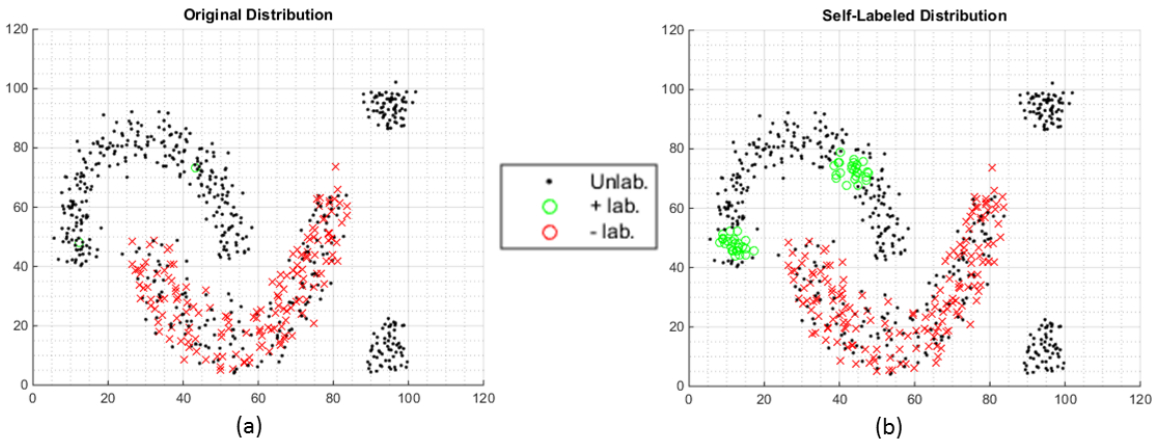


Figure 4.1: Self-Training with I.W. N.N. on 2D Toy Data. The figure in (a) displays the input, and the figure in (b) displays the labels of the instances after self-training.

training method's performance with precision. Note that the positive instance count after self-training is always 15. In the figure, we present results for each class along with the different color codes for different annotated sketch counts. Even tough this method can be considered conservative, it still labels many negative instances as positive. The falsely labeled instances pose problem for the learner in the final step. We present examples that are labeled as positive by instance-wise nearest neighbor method in Table 4.1. In the table, mislabeled instances are painted by red and correctly labeled instances are painted with green.

4.2 Mean of Distances Nearest Neighbor

Mean of distances nearest neighbor self-training method aims to label unlabeled instances that have the lowest mean distance to all of the positive labeled instances. We can interpret the operation of this method as finding a mid-point between all of the positive instances, and labeling the unlabeled instances that are closest to this mid-point, towards the positive instances. Operation of this method is represented using 2D toy data in Figure 4.3. Note that the positive instance set size was two before the method labeled 50 other instances. As can be seen, different from the

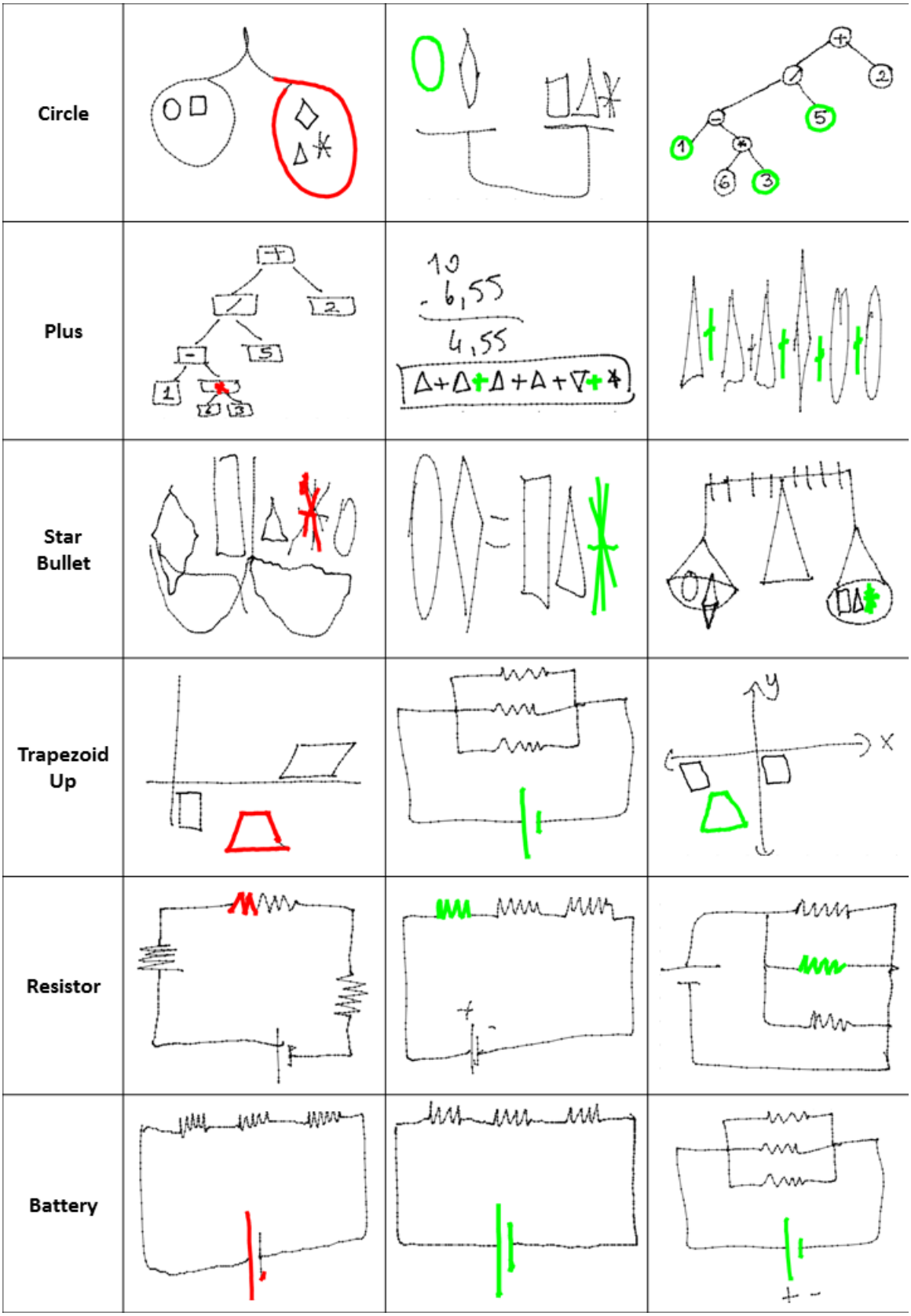


Table 4.1: Examples Labeled by I.W. N.N. Self-Training

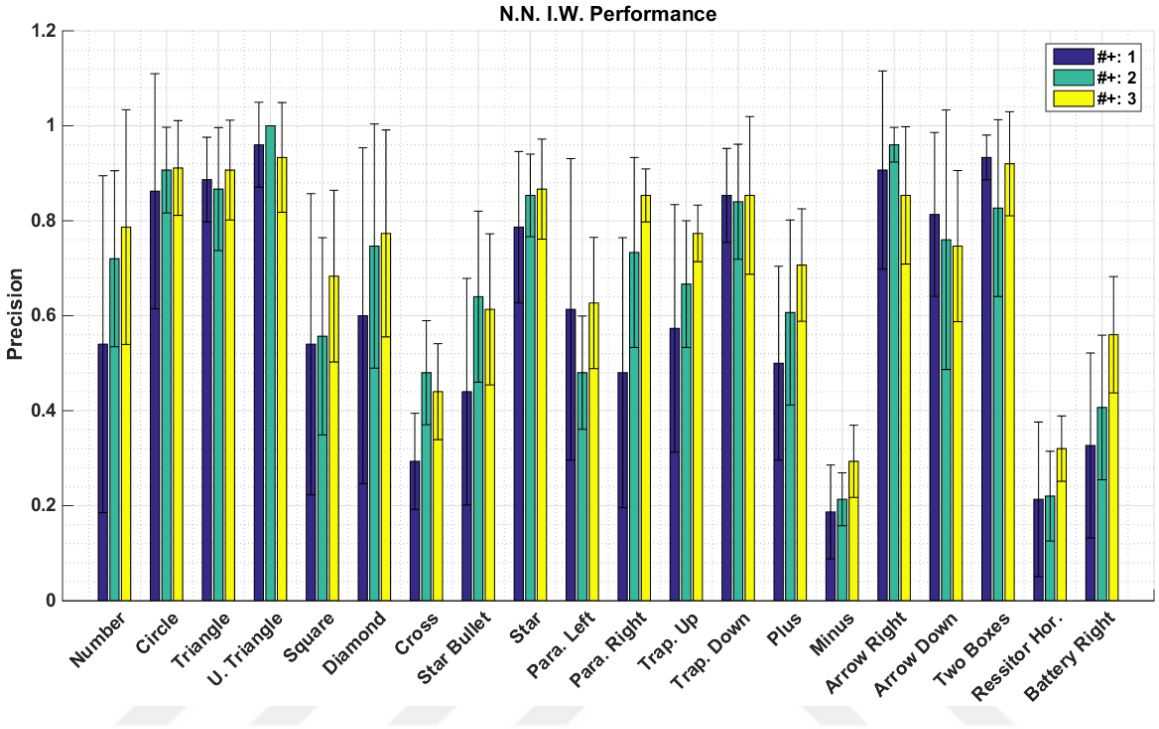


Figure 4.2: Performance of I.W. N.N. Self-Training

nearest neighbor instance-wise self-training, this method labels instances to "fill the gap" between the positive instances. We present examples that are labeled as positive by mean of distance nearest neighbor method in Table 4.2. In the table, mislabeled instances are painted by red and correct labeled instances are painted with green.

In Figure 4.4, we present the performance of mean of distance nearest neighbor method. We use the precision as performance measure. We see similar results with instance wise nearest neighbor method.

4.3 Artificial Instance Generation

Formally, artificial instance generation is not a self-training method. While self-training methods extend the labeled set via labeling instances from unlabeled instances, artificial instance generation method extends the labeled set via generating novel artificial instances. In order to generate artificial instances, we apply linear

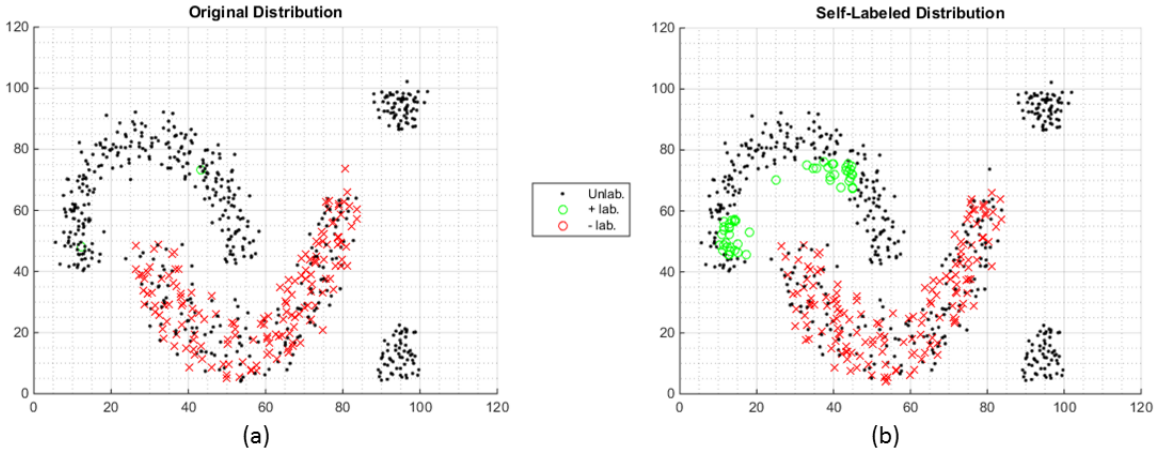


Figure 4.3: Self-Training with Mean of Distance N.N. on 2D Toy Data. The figure in (a) displays the input, and the figure in (b) displays the labels of the instances after self-training.

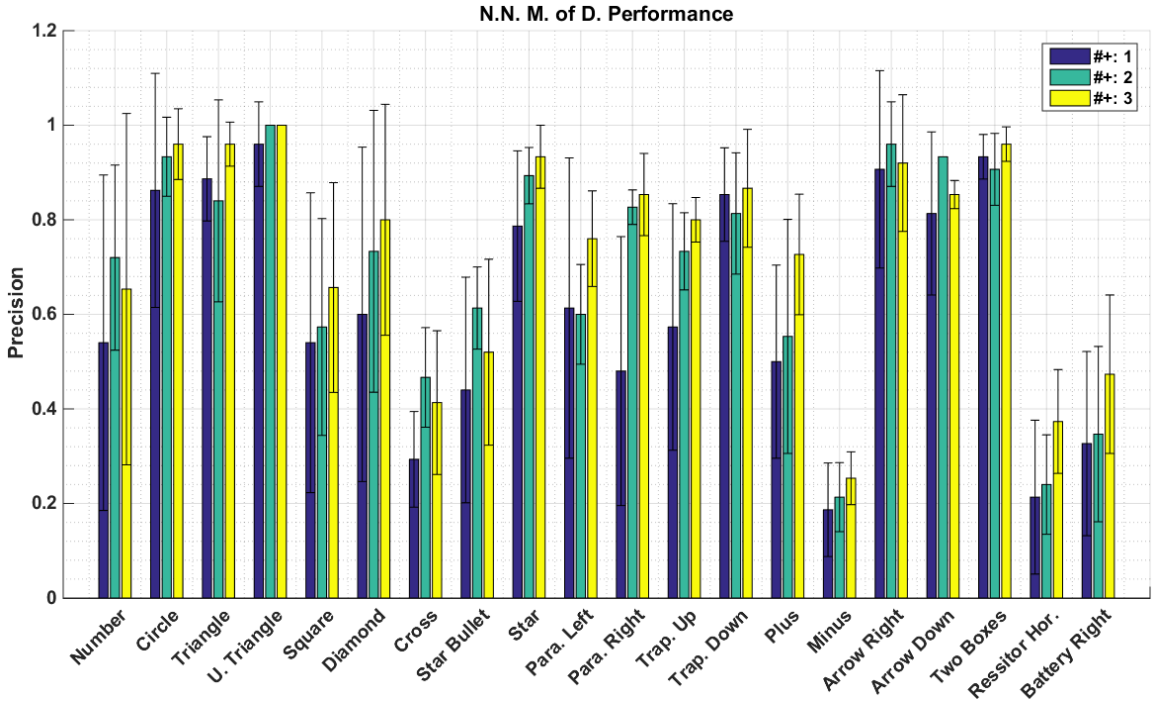


Figure 4.4: Performance of Mean of Distance N.N. Self-Training

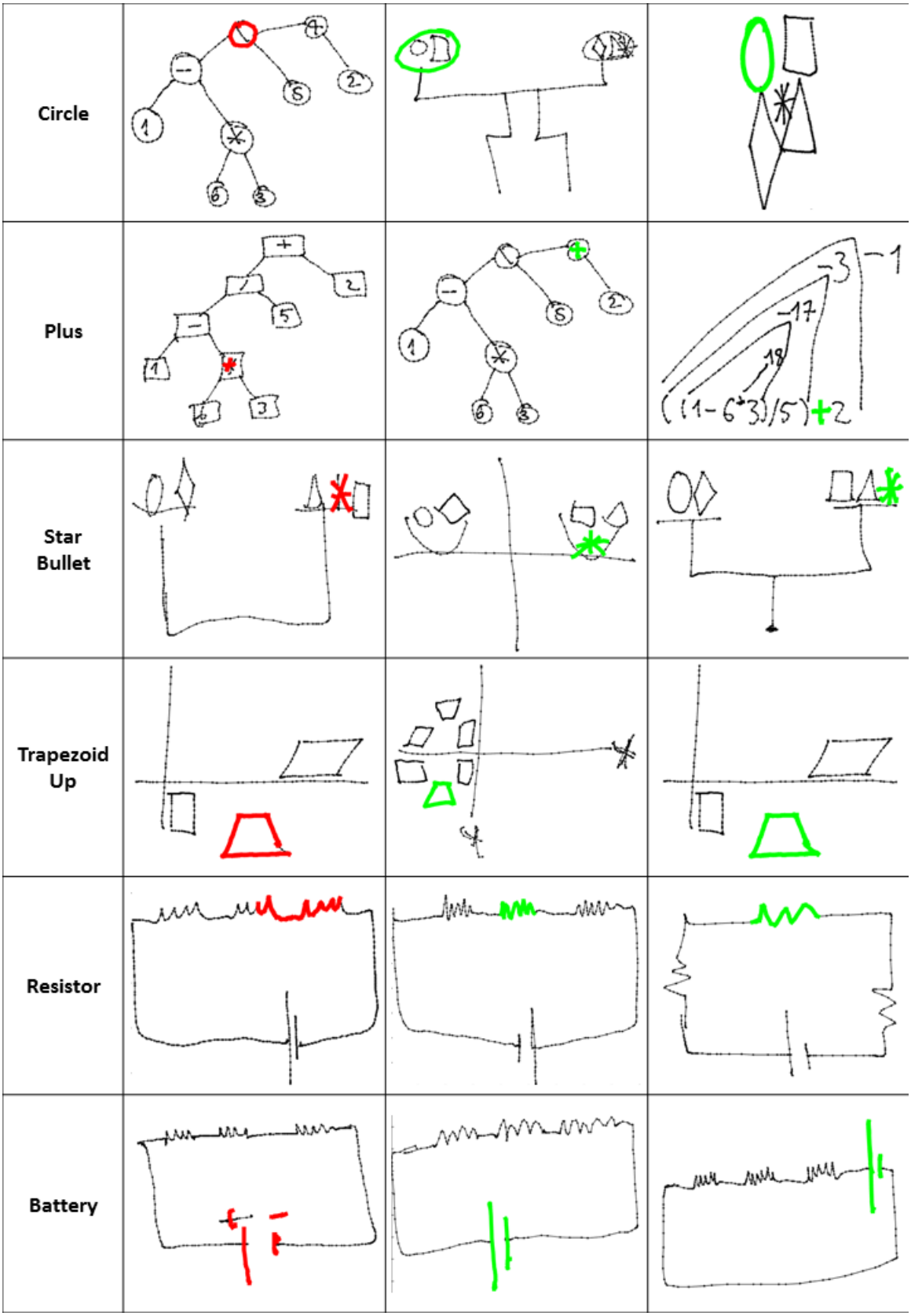


Table 4.2: Examples Labeled by Mean of Distance N.N. Self-Training

geometric transformations on positive labeled instances. We use two transformations in this study: rotating and shearing. In our study, we generate 10 novel instances for each positive labeled instance. For instance, if there are two positive labeled instances, we generate 20 new instances which result in total of 22 positive instances.

We use only shearing and rotation while generating new instances as other transformations, like mirroring, may generate sketches that does not have the same class with the original sketch object (i.e. mirroring a triangle on x-axis generates an upside-down triangle). While generating, we apply transformations in random order. In our experiments, we select the free parameters of transformations randomly for each generation within range 0' - 15' for rotation, and 0 - 0.25 for shearing.

This method has strong connection with the work of Miller et. al.. In their work, they define the process called congealing which learns the transformation parameters from a data set and generate synthetic instances using this information from single example. Although they offer a method to develop a recognition method that operates with a single example, their method utilize transformed information from the data set that has many instances of the same class. In our work however, we do not utilize transformed information or any other kind of data apart from the data set visible to the system.

In Table 4.3, we present some examples of instances that are generated by this method.

4.4 Context Based Self-Training

People tend to follow certain placement patterns while sketching. For instance, while drawing serial circuit diagrams, people tend to draw the components on the same line. Another example can be stick figure drawing. It is almost guaranteed that the head of the stick figure will be placed on top of the body. There are many causes for placement patterns to occur. It may because of the structure of the object, the syntax for diagrams, or because of the effort to make the sketch spruce. Even tough we did not study the reason of the placement patterns, we realized that the placement

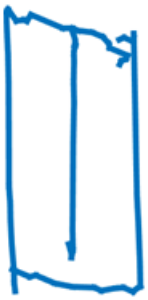
	Diamond	Double Box	Square
Original			
Generated			
Generated			
Generated			
Generated			

Table 4.3: Examples from the Instances Generated by A.I.G.

patterns are common in sketches.

Utilizing placement patterns aid self-training for sketch recognition in multiple aspects. Firstly, we have more diversity in the labeled instance set. The diversity in the labeled set is important as the labeled instance set carries more information when it embodies more diverse instances. Secondly, we have a better precision. It is because we introduce additional relevant properties for each instance when we use placement patterns along with the appearance.

We establish a score based approach for utilizing placement patterns along with the appearance. In our approach, we calculate two different scores for each instance: context score and appearance score. Context score represents how well an instance fit to the common placement pattern. Appearance score represents the visual similarity between an instance and other positive labeled instances. With those two scores, we can select the unlabeled instances that excel in terms of appearance and the placement for labeling. In order to implement this approach, we propose a novel five-step self-training algorithm. The consecutive steps follows as clustering, prediction, score calculation, score scaling, and instance selection.

Even tough placement patterns are common, sketches may exhibit different placement patterns. In order to select and operate on the sketches that exhibit the same pattern, we cluster sketches based on their appearance and select the most populated cluster to operate on. The sketches that are not members of the most populated cluster are dismissed. In our experiments, we cluster sketches into 20 groups using hierarchical clustering.

To extract placement patterns in a sketch, the places of sketch objects are required. In order to reveal the positions of the sketch objects, we predict the classes of each instances for each sketch. In this step we practice multi-class classification. Instances that are predicted as members of the positive class (annotated class) are considered as sketch object candidates in later steps of the algorithm.

In order to model the placement patterns, we analyze each sketch object candidate independently. In order to implement placement information for each sketch object

candidate, we pair them with the other sketch objects placed in the same sketch. For each pair, we calculate the distances between objects, the angle between objects, and the classes of the paired sketch objects. We can illustrate this implementation as a list that has three columns. Each column represents one of the component we calculated and each row represents a pair. We limit the number of pairs a sketch object candidate can have to three. This means that we generate more than one model for a sketch object candidate. For instance, if there are two other sketch objects other than the sketch object candidate in a sketch; we generate two models that have one pair, and one model that have two pairs.

To calculate the context score of a candidate, the candidate's model is compared with the guide models. We define guide models as the placement pattern models that are extracted from the annotated sketches. We treat the guide models like templates which we expect all the patterns in all the sketches be similar with.

Two placement pattern models must share the same class formation in order to be compared. For instance, if one model has more pairs or the classes of paired sketch objects does not match with the other model, comparison is not possible. While calculating context scores, we calculate the differences of the distances and differences of the angles the pairs that have the same class formation. After calculation, we average and sum the distance and angle differences to yield single value that represents the context score. If there are multiple guide models that share the same class formation with the candidate's model, we calculate the scores for all models and accept the highest score.

Appearance score is calculated as the distances in feature space between the candidate and the instances which label is inherited. After context and appearance scores are calculated for all candidate sketch objects, the scores are scaled and normalized. We realized that both context and appearance score exhibit a sigmoid shaped distribution. Thus, we fit sigmoid functions for both distributions to scale scores to have values between zero and one [Platt et al., 1999]. After scaling, we combine context and appearance scores to have a single score for each candidates. We combine two scores us-

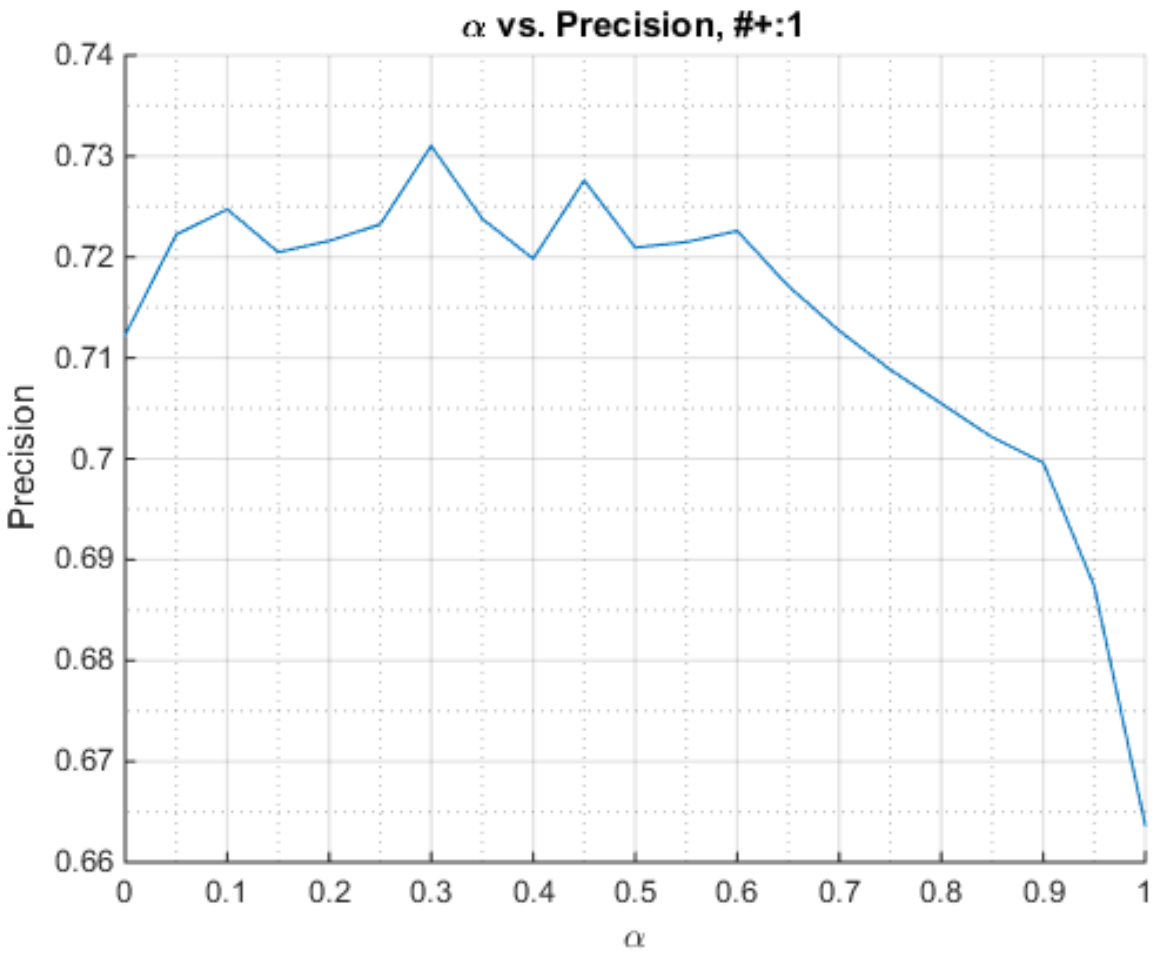


Figure 4.5: Change in Performance with α

ing a linear combination where a trade-off parameter controls the weight between the context and appearance. We use $Score_{Final} = Score_{Context} * \alpha + Score_{Appearance} * (1 - \alpha)$ formula to calculate the final score, where α is the trade-off parameter. After the calculation of final scores, we pick the candidates for positive labeled instance set that have the highest scores.

We present the performance change with α parameter in Figure 4.5. As can be seen, we achieve better results when α value is set around 0.3. This means context information aids to have better precision when combined with appearance.

We present examples that are labeled as positive by context based self-training in

Table 4.4. In the table, mislabeled instances are painted by red and correct labeled instances are painted with green. Note that some of the false positives are occurred because of the high match in pattern placement.

Our method have strong similarities with the method Song et. al. developed recently [[Song et al., 2016]]. However, our method is based on scoring the instances for self-learning and can perform in any context and in any sketch type; where method Song et. al. developed is specialized for clock drawings and is used for learning with conditional random fields.

4.5 Comparing Self-Trainers

In this section, we compare self-trainers in terms of precision and diversity. We present precision results of self-trainers obtained for single annotation setting in Figure 4.6. As can be seen, while instance-wise and mean of distance nearest neighbor algorithms achieve similar performances, context based method achieves superior performance.

We compare diversity using a computational approach. For each experiment, we find the smallest hyper-sphere that contains all positive instances the data set have after self-training. We use linear kernel support vector data description to find the hyper-sphere [Tax and Duin, 2004]. The radius of the hyper-sphere is an indicator of diversity. If the radius is large, it means that the instances are distributed in a larger space, containing more information, and vice-versa. In Figure 4.7, we present the sorted pairwise differences of the radiuses. In this figure, we see that context based method have larger radius than other methods, means it achieves to label more diverse instances.

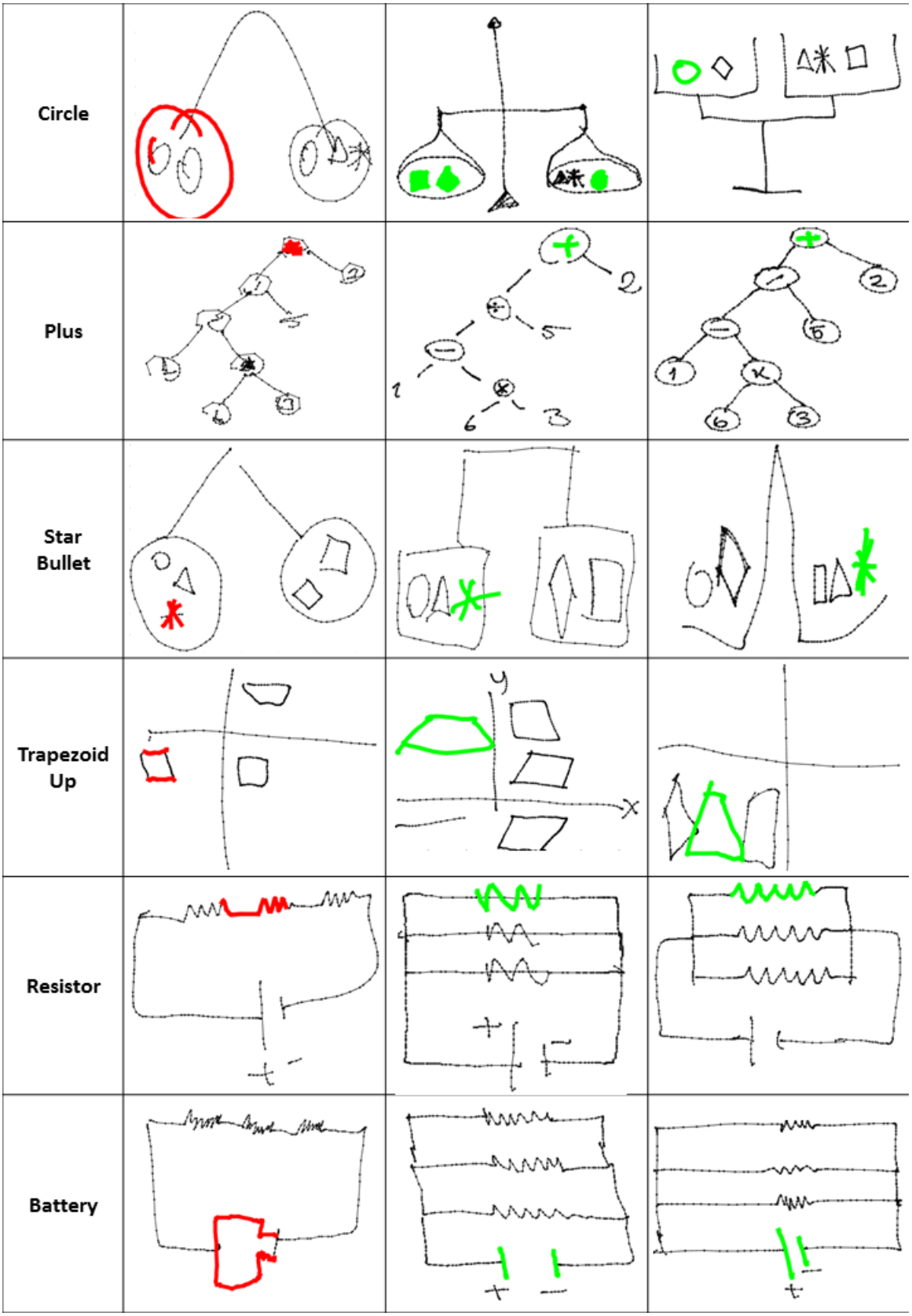


Table 4.4: Examples Labeled by Context Based Self-Training

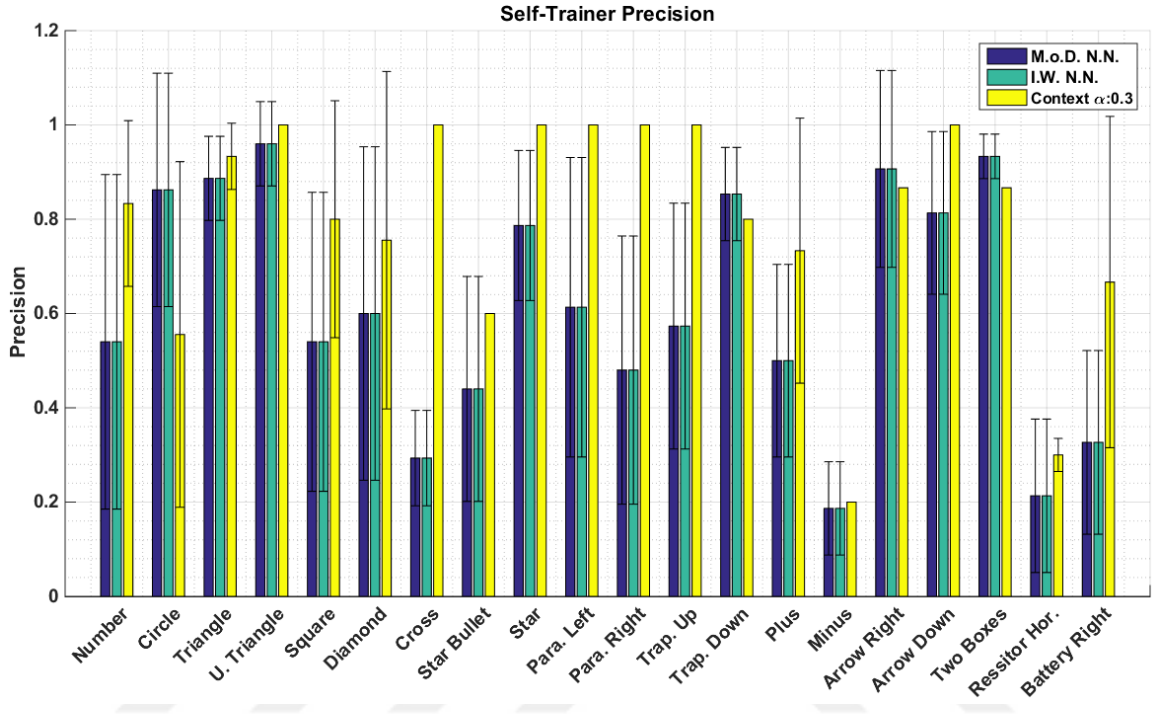


Figure 4.6: Precisions of Self-Trainers

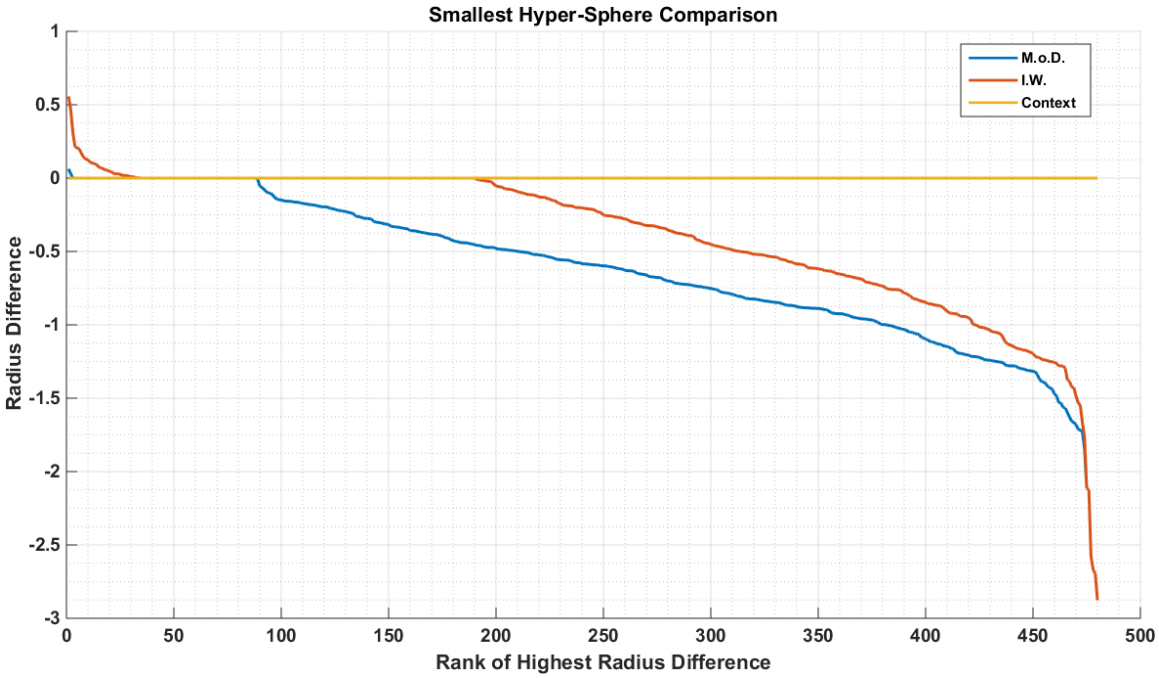


Figure 4.7: Smallest Hyper-Spheres Radius Differences of Self-training Methods

Chapter 5

RESULTS AND CONCLUSION

5.1 Fuzzy Evaluation

The fragment and combine approach comes with a problem. When a stroke is over-fragmented, it introduces instances that are very similar to each other. When a stroke is part of a positively annotated sketch piece and is over-fragmented, it generates instances that are very similar to the positive class, but labeled as negative in ground truth. In this condition, even if an instance is a good example of the positive class and predicted as positive by a classifier, it will be considered as false positive.

It is specious to judge a recognizer by direct comparison with ground truth annotations when there is over-fragmentation issue in the data set. We evaluate recognizers using fuzzy comparison in order to prevent the effect of over-fragmentation in our results. With fuzzy comparison, we accept a positive prediction as true positive if the predicted instance share sketch points with an annotated instance in the same sketch more than a threshold. The comparison is made by comparing sketch points. An illustration of sketch pieces comparison is presented in Figure 5.1. In this figure, the first row presents the instances with sketch points indicated by yellow circles. The first column represents the instance that is annotated as positive. Other columns represent some of the sub-parts of the annotated instance. The matching percentages and the number of sketch points matched are indicated for each instance. For example, the instance in the third column has 85% match with the annotated instance. This means if the threshold is set to a value below 85%, this instance will be considered as true positive. In order to prevent multiple true positives for the same annotated instance, we accept only the first true positive detection for an annotated instance, and naturalize other true positives for that annotated instance. In our experiments,

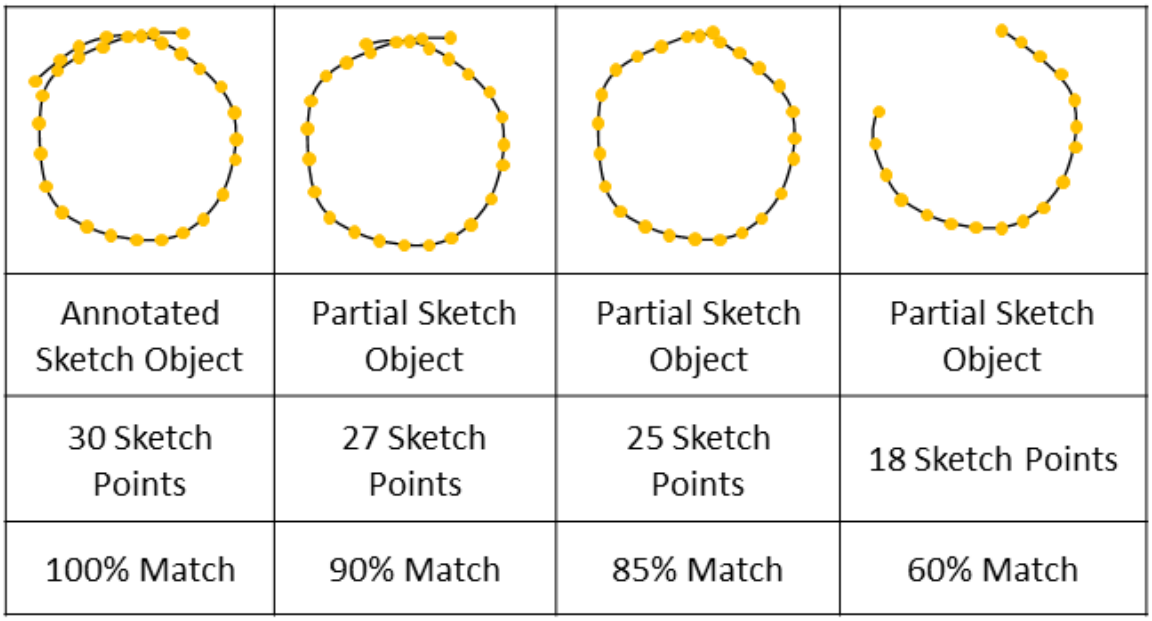


Figure 5.1: An Illustration of Matching Between Sketch Parts

we set the threshold to 90%.

5.2 Overall Results

We set nearest neighbor without self-training as our baseline. Even tough this is an ad-hoc method, it performs well in our setting. Moreover only a few classifier in the literature can work with such low number of instances. Its high performance can be linked to the strong assumption it makes on the smoothness of the feature space thus its independence of the shape of the data distribution. Nearest neighbor classifier's superior performance on data sets with low labeled instance size is also reported by other researches in different studies [Tax, 2001].

In this section, we present the final performances of combined methods. The final performances are affected by numerous factors: (1) number of sketches annotated, (2) performance of conservative rejection, (3) performance of self-training, and (4) performance of learners. Errors made by one of the methods accumulate and may cause the final performance to be lower. We present the final performances with

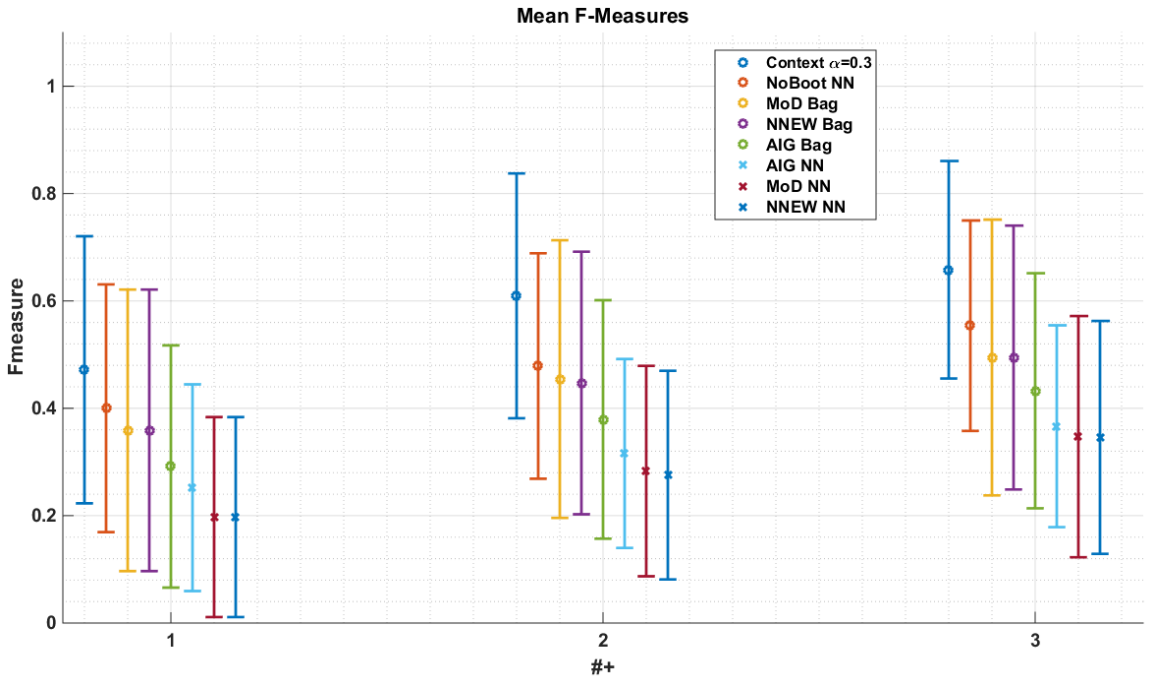


Figure 5.2: Overall Mean Performances

F-measure. The reason of using f-measure is the imbalance of the data set.

In Figure 5.2, we present the overall performances with reporting mean and standard deviations for different classes. In Figure 5.3, we present the pairwise differences of method performances with the nearest neighbor method. While producing this figure, we calculated the differences of each experiments that have the same setting. As we designed the experiments with repeated measure, we can take the differences without additional analyses.

As can be seen from the results, our novel context based self-training method with linear SVM bagging exhibits superior performance when compared to other methods. We link the superior performance of context based self-training to its higher precision and diverse instance selection capability. We see the expected performances increase as the number of annotated sketches increases. From results, we see that self-training may hinder learning with submitting negative instances as positive in our setting. This is a common problem in self-training but is prevented with utilizing context

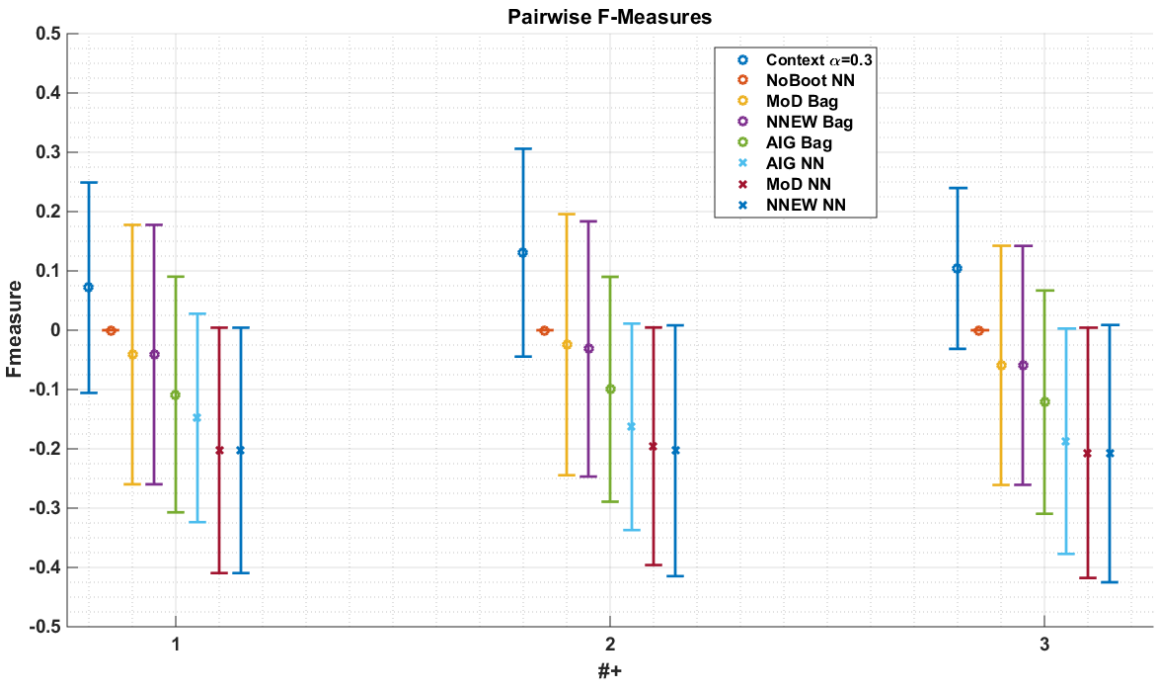


Figure 5.3: Overall Pairwise Performances

information.

5.3 Conclusion

In conclusion, we analyzed and presented results for different methods for sketch recognition with few examples. We used our realistically collected scene sketch data set while analyzing the methods. Results show that our novel context based self-learning outperforms our baseline nearest neighbor with no self-training. Even tough we set nearest neighbor without self-learning as baseline, it performs better than other methods except the context method. The conservative rejection method exhibit well performance for scaling down the large data sets. Along the classifiers, bagging is a more successful method when compared to nearest neighbor. Finally, we show that self-training without utilizing context information hinders the learning for sketch recognition within our setting.

5.4 Future Work

We believe that our work has room for further improvements. All of the steps in our pipeline can be further studied independently. Apart from the methods we examined in this work, there are many methods proposed in literature which can be used to achieve better results in our system. For future work for sketch recognition with few example, we point two major directions.

One direction to study is active learning for sketch recognition with few examples. In our current system, the user annotates sketches selected in random. However, it is possible to increase performance rates both for self-training and classification if sketches to be annotated are chosen to have better performances. The work of Yanik et. al. on active learning for sketch recognition pose a guideline for such future work.

Even tough we utilize context information only for self-training, it can also improve the classification rates. As future work, it is worth examining the use of context information for classification. The work of Song et. al. on using context for sketch recognition pose a guideline for such future work.

BIBLIOGRAPHY

[Chawla et al., 2002] Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P. (2002). Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357.

[Cortes and Vapnik, 1995] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

[Douglas and Peucker, 1973] Douglas, D. H. and Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122.

[Eitz et al., 2012] Eitz, M., Hays, J., and Alexa, M. (2012). How do humans sketch objects? *ACM Trans. Graph.*, 31(4):44–1.

[Fei-Fei et al., 2006] Fei-Fei, L., Fergus, R., and Perona, P. (2006). One-shot learning of object categories. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(4):594–611.

[Fei-Fei et al., 2007] Fei-Fei, L., Fergus, R., and Perona, P. (2007). Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59–70.

[Gentzsch, 2001] Gentzsch, W. (2001). Sun grid engine: Towards creating a compute power grid. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 35–36. IEEE.

[Jain and Chandrasekaran, 1982] Jain, A. and Chandrasekaran, B. (1982). [a]dimensionality and sample size considerations in pattern recognition practice, [o] handbook of statistics. pr krishnaiah and ln kanal, eds., vol. 2.

[Miller et al., 2000] Miller, E. G., Matsakis, N. E., and Viola, P. A. (2000). Learning from one example through shared densities on transforms. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 1, pages 464–471. IEEE.

[Ouyang and Davis, 2009] Ouyang, T. Y. and Davis, R. (2009). A visual approach to sketched symbol recognition.

[Platt et al., 1999] Platt, J. et al. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74.

[Raudys and Jain, 1991] Raudys, S. J. and Jain, A. K. (1991). Small sample size effects in statistical pattern recognition: recommendations for practitioners. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (3):252–264.

[Salperwyck and Lemaire, 2011] Salperwyck, C. and Lemaire, V. (2011). Learning with few examples: An empirical study on leading classifiers. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 1010–1019. IEEE.

[Skurichina and Duin, 1996] Skurichina, M. and Duin, R. P. (1996). Stabilizing classifiers for very small sample sizes. In *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, volume 2, pages 891–896. IEEE.

[Song et al., 2016] Song, Y., Davis, R., Ma, K., and Penny, D. L. (2016). Balancing appearance and context in sketch interpretation. *arXiv preprint arXiv:1604.07429*.

[Tax, 2001] Tax, D. M. (2001). *One-class classification*. TU Delft, Delft University of Technology.

[Tax and Duin, 2004] Tax, D. M. and Duin, R. P. (2004). Support vector data description. *Machine learning*, 54(1):45–66.

[Tirkaz et al., 2012] Tirkaz, C., Yanikoglu, B., and Sezgin, T. M. (2012). Sketched symbol recognition with auto-completion. *Pattern Recognition*, 45(11):3926–3937.

[Triguero et al., 2015] Triguero, I., García, S., and Herrera, F. (2015). Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study. *Knowledge and Information Systems*, 42(2):245–284.

[Tumen et al., 2010] Tumen, R. S., Acer, M. E., and Sezgin, T. M. (2010). Feature extraction and classifier combination for image-based sketch recognition. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, pages 63–70. Eurographics Association.

[Viola and Jones, 2001] Viola, P. and Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511. IEEE.

[Wolf and Martin, 2005] Wolf, L. and Martin, I. (2005). Robust boosting for learning from few examples. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 359–364. IEEE.

[Yanık and Sezgin, 2015] Yanık, E. and Sezgin, T. M. (2015). Active learning for sketch recognition. *Computers & Graphics*, 52:93–105.

[Zhou et al., 2007] Zhou, Z.-H., Zhan, D.-C., and Yang, Q. (2007). Semi-supervised learning with very few labeled training examples. In *Proceedings of the national conference on artificial intelligence*, volume 22, page 675. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

[Zhu, 2005] Zhu, X. (2005). Semi-supervised learning literature survey.