<table>
<thead>
<tr>
<th>Title</th>
<th>ActiveCrowd: integrating active learning with Amazon Mechanical Turk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author(s)</td>
<td>Chi, Fung Cheung (池鳳翔)</td>
</tr>
<tr>
<td>Citation</td>
<td>Chi, F. C. (2015). ActiveCrowd: integrating active learning with Amazon Mechanical Turk (Outstanding Academic Papers by Students (OAPS)). Retrieved from City University of Hong Kong, CityU Institutional Repository.</td>
</tr>
<tr>
<td>Issue Date</td>
<td>2015</td>
</tr>
<tr>
<td>URL</td>
<td><a href="http://hdl.handle.net/2031/8303">http://hdl.handle.net/2031/8303</a></td>
</tr>
<tr>
<td>Rights</td>
<td>This work is protected by copyright. Reproduction or distribution of the work in any format is prohibited without written permission of the copyright owner. Access is unrestricted.</td>
</tr>
</tbody>
</table>
ActiveCrowd: Integrating Active Learning with Amazon Mechanical Turk

Chi Fung Cheung

City University of Hong Kong
2015
Acknowledgement

I would like to express my deepest appreciation to Dr. Sarana Nutanong, my final year project supervisor. His patient guidance and enthusiastic encouragement throughout the project are the keys to the successful completion of this project. I thank Miss. Conny Yau, an instructor from the English Language Centre, for her language support on report writing. My thanks also go to Mr. Peter Xu for his help in verifying framework installation procedures and code documentations.

In addition, special thanks go to Amazon Web Services support staff for their help in handling security issues of using Amazon Mechanical Turk.
Abstract

Machine learning is a technique that builds classification and prediction models through learning from samples. It is proven to be useful in scientific research such as DNA pattern recognition and climate modeling [1 & 2]. It is also adopted in many real-life applications, including spam filtering, image searching and optical character recognition (OCR) [3, 4 & 5]. Theoretically, the more samples being provided to a learning model, the more accurate the model can be. However, supervised learning requires that samples be provided along with their labels, which can be expensive to obtain in terms of the human power required for labeling tasks [6]. It greatly hinders the adoption of machine learning in resource-limited environments.

Meanwhile, crowdsourcing allows requestors to obtain scalable workforce and services from a large crowd of people. Amazon Mechanical Turk [7] (MTurk) is one popular online crowdsourcing platform which enables requestors to publish requests to more than 500 thousand registered workers. It has potential to solve the problem of sample labeling [8], but so far no integration of machine learning and crowdsourcing is implemented in a way that can serve general machine learning purposes.

In this project, a machine learning framework named ActiveCrowd was designed and implemented to allow anyone who has basic programming knowledge to build machine learning model for general purposes. The framework adopted active learning technique and integrated scikit-learn [9], which is a superior machine learning library written in Python [10] and published under BSD license, with Amazon Mechanical Turk as the label annotator in a low cost and efficient manner. The framework is able to reduce the implementation effort required for building machine learning models and makes the supervised learning process completely automated.
# Table of contents

Acknowledgement.................................................................................................................... 2

Abstract ........................................................................................................................................ 3

1. Introduction ................................................................................................................................. 9

   1.1 Overview ................................................................................................................................. 9

   1.2 Applications of machine learning .......................................................................................... 9

       1.2.1 Applications in scientific research ................................................................................ 9

       1.2.2 Application in commercial environment ......................................................................... 10

       1.2.3 Application in other information technologies ............................................................. 11

   1.3 Problems existing in the current situation ............................................................................. 12

       1.3.1 Means of acquiring sample-label pairs .......................................................................... 12

       1.3.2 Technical knowledge and implementation effort required .......................................... 13

   1.4 Existing solutions and their limitations .................................................................................. 14

       1.4.1 Dedicated applications .................................................................................................... 14

       1.4.2 Machine learning application development frameworks ......................................... 14

       1.4.3 Integration of crowdsourcing and supervised learning ................................................. 16

   1.5 Objectives and Scope ............................................................................................................. 17

2. Literature Review ...................................................................................................................... 18

   2.1 Machine learning categories ................................................................................................. 18

       2.1.1 Unsupervised learning ................................................................................................... 18

       2.1.2 Supervised, semi-supervised and active learning ....................................................... 18

       2.1.3 Reinforcement learning ................................................................................................ 20

   2.2 Machine learning output models ........................................................................................... 20
4.1.1.2 MTurk Record table ................................................................. 35
4.1.1.3 MTurk Failure table ............................................................... 36
4.1.1.4 MTurk Answer table ............................................................... 36
4.1.2 MTurk Connection ................................................................... 37
4.1.3 HIT creation ............................................................................ 38
  4.1.3.1 Properties file ..................................................................... 38
  4.1.3.2 Question file ...................................................................... 40
  4.1.3.3 Input file ........................................................................... 44
  4.1.3.4 Success file ....................................................................... 47
4.1.4 HIT management ...................................................................... 47
  4.1.4.1 Checking HIT status ........................................................... 47
  4.1.4.2 Retrieving answers ............................................................ 48
  4.1.4.3 Removing HITs ................................................................. 49
4.2 Learning automation ................................................................... 50
  4.2.1 Database designs for learning automation ............................... 50
    4.2.1.1 Sample table ................................................................. 50
    4.2.1.2 Execution record table .................................................. 50
  4.2.2 Initialization of learning ........................................................ 51
  4.2.3 Sampling strategy .................................................................. 52
    4.2.3.1 Making prediction on unlabeled samples ....................... 53
    4.2.3.2 Calculating entropy ....................................................... 53
    4.2.3.3 Selecting samples ......................................................... 54
  4.2.4 Accuracy evaluation ............................................................... 56
4.2.4.1 K-Fold Cross-Validation ......................................................... 56
4.2.4.2 Control Group Validation ......................................................... 58

4.3 Project and client application creation and execution .................. 59

4.3.1 Database design for project and client application ................. 59
4.3.1.1 Project table ................................................................. 59
4.3.2 Project Creation .................................................................. 60
4.3.3 Client application creation pre-requirements ......................... 61
4.3.3.1 Active learning and MTurk setting .................................. 61
4.3.3.2 Scikit-learn setting ......................................................... 62
4.3.3.3 User function setting ....................................................... 64
4.3.4 Client application creation .................................................... 65
4.3.5 Client application execution .................................................. 67
4.3.5.1 Uploading samples ......................................................... 67
4.3.5.2 Execution ................................................................. 68

4.4 Web application interface .......................................................... 69
4.4.1 Structure of the web application (server-side) layer ............... 69
4.4.1.1 General request control ............................................... 71
4.4.1.2 General template layout ............................................. 71
4.4.2 Active learning performance and MTurk monitoring .......... 73
4.4.2.1 Line chart for Active learning performance monitoring .... 73
4.4.2.2 Pie charts for sample distribution and MTurk performance monitoring ................................................................. 78

5. Results ....................................................................................... 83
5.1 Labeled samples amount and model accuracy ........................................ 83
  5.1.1 Experiment design ........................................................................ 83
  5.1.2 Experiment on the optical recognition of handwritten digits data set ................................................................. 84
    5.1.2.1 Experiment results .............................................................. 85
    5.1.2.2 Evaluations and discussion ............................................. 87
  5.1.3 Experiment on the spam SMS data set .................................... 89
    5.1.3.1 Experiment results .......................................................... 90
    5.1.3.2 Evaluations and discussion ............................................. 92
  5.1.4 Brief summary ......................................................................... 93
  5.2 Effort required to obtain a classifier ........................................... 95
    5.2.1 Comparison between the framework and alternatives ........... 95
    5.2.2 Brief summary ................................................................... 96
  6. Conclusion .................................................................................... 96
    6.1 Achievements .......................................................................... 96
    6.2 Future improvement .................................................................. 97
  7. Monthly logs ............................................................................... 99

References ..................................................................................... 103
Appendix I: List of charts ................................................................. 111
Appendix II: List of tables ................................................................. 112
Appendix III: List of figures ............................................................... 114
1. Introduction

1.1 Overview
Machine learning is a technique that enable machine to learning from a set of given data. The goals of performing machine learning could be to identify the pattern of the given dataset, build models to enable data classification, or even to create prediction models to forecast future phenomenon or events. Besides being used with other information technologies, machine learning is also widely adopted in scientific research and commercial environments. However, making use of machine learning requires a long study time. Also, developing a learning applications requires considerable implementation effort. More importantly, many environments require users themselves to do sample labeling which need huge manual effort. In order to solve the above problems and thus making organizations or individuals who have limited skills and resources can also benefit from machine learning, the ActiveCrowd framework is designed and developed. It integrates active learning with crowdsourcing to achieve automated machine learning and free users from heavy labeling tasks in a cost efficient manner. In addition, it greatly shortens the study time and reduces development effort by allowing creation and execution of learning applications to be done via its graphical user interface.

1.2 Applications of machine learning

1.2.1 Applications in scientific research
Machine learning is widely adopted in different aspects. One major field is scientific exploration. Much research relies on analyzing huge amount of data
collected by scientific instruments or generated during computer simulations [11, 12 & 13]. With the advancement of science and technology, new instruments can capture data with extraordinary frequency and precision and results in exponential growth of data volume [14 & 15]. Traditionally, scientists analyze the retrieved data with data analysis tools, which require scientists to design analysis strategy and perform data management by themselves. The workflow heavily requires manual labor from scientists. In fact, in many cases, these scientists work like information technology professionals with expertise in computer data analysis tools [14, 15 & 16]. This greatly hinders them from focusing on their own research.

Unlike traditional data analysis tools which can only follow established instructions explicitly, machine learning can discover samples’ pattern and build models based on them. As a result, scientists no longer need to design their own analysis instructions for the tools but still can obtain promising results. Machine learning is applicable in almost all scientific fields. For example, in biology and medical science, scientists make use of the pattern recognition and classification ability of machine learning to create solutions to identify hereditary disease and virus [17, 18, 19 & 20]. Similarly, chemists use machine learning to create classification models to identify chemical compositions and structures of given samples [21, 22 & 23].

1.2.2 Application in commercial environment

In a commercial environment, machine learning is especially useful in the meaning of making prediction. In a stock market, the price of a company is greatly related to the market behaviors and news, which can be represented in the form of data. Much research has been done and has proven that machine learning can achieve more than 70% prediction accuracy in the stock market [24 & 25]. With enough historical data and computing power provided, machine learning can reach 85% accuracy in next-day prediction [24 & 25]. Some big
firms already adopted commercial machine learning systems in order to predict the market performance and changes, or even perform automated trading [26]. Recommender System is another significant application of machine learning for commercial purposes. In order to obtain the greatest profit, retailers usually try to provide customers the information of products which they are most likely to be interested in. It requires a system that can identify the preferences of an individual customer and predict their rating to other products. Machine learning is one of the main approaches currently adopted to provide highly accurate recommendation. For example, the Pandora Internet Radio uses a content-based recommender system, which is an application of machine learning, to recommend music to users [27].

Besides financial applications, machine learning can also be applied in infrastructure and benefit the society. For instance, many papers suggested that machine learning should be used in power grids to improve their reliability by passing historical data to machine learning algorithms to forecast the need for maintenance [28 & 29].

1.2.3 Application in other information technologies

Machine learning technology can be used with other information technologies to provide better services to users. In spam filtering services, machine learning algorithms treat emails as sample input with “spam” or “not spam” as labels. Models learn the pattern of email contents that were marked as “spam” by a user so the spam filtering system can classify emails according to individual user’s behaviors without the need to explicitly specify rules or definitions of spam emails [30 & 31].

In search engines, machine learning is used to achieve image searching with high matching accuracy. With abundant amount of labeled images provided to learning models, search engine system can classify the content of a given image and then further retrieve image with similar signature from the database [4 & 32].
Optical character recognition (OCR) technique uses similar approaches to transform text images into text, or perform handwriting recognition [5 & 33].

1. Problems existing in the current situation

1.3.1 Means of acquiring sample-label pairs

Machine learning enables machines to learn from huge amount of samples. For building classification and prediction models, samples need to be provided along with corresponding labels in order to show the differences of samples to learning models [8 & 34]. Nowadays, instruments and systems can capture data with extraordinary frequency and precision so users can obtain large amount of samples easily at low cost. However, in many environments, labels cannot be generated or gathered along with samples. For example, a human protein-protein interaction experiment can generate hundreds of protein pairs but the type of pair interaction are not known [35]. In commercial world, many companies have many scanned copies of paper receipts needed to be transform to digital records [36]. In mechanical engineering, robot execution failures data record the detailed states of a robot before failures but do not tell the exact reason of causing failures [37]. Moreover, labels cannot be easily deduced from samples using static program instructions. That is why users want to use machine learning to build models to achieve classification and regression. As a result, before the learning process could be started, users themselves have to play the role of annotators and undertake the labeling tasks.

Although active learning can be adopted to maximize the efficiency of learning, and thus reduce the amount of samples required, there is still a considerable weight of manual effort required from users [8 & 34]. An authentic example happened to Zhu [23] during the training of a model for speech recognition. In his training process, annotation of utterances at word-by-word level took 10 times longer than the audio samples, while annotation at phoneme level took 400 times longer than the actual speech. In other words, it took more than 6
hours to label a one-minute speech. The labeling tasks even required expertise in linguistics to create accurate labels. Even if the labeling jobs are very simple to human, building a scientifically accurate model requires numerous iterations which involves enormous amount of samples being selected to be labeled [8].

In addition, sample labeling is required in every iteration but the time needed for completing an iteration might vary. Labeling tasks might pop-up from time to time and greatly disturb users from doing other works. All these eventually turn into huge manual effort, taking a lot of precious time from users and reducing their concentration, working efficiency and accuracy in other subjects.

1.3.2 Technical knowledge and implementation effort required

Although machine learning is widely adopted in different fields such as scientific research, commercial environment and other information technologies, it is observable that most of the uses of machine learning are limited to large-scale applications which were developed by large organizations. There are several resistances to the popularization of machine learning among small and medium-sized companies (SMC), as well as small-scale scientific research with limited funding.

Stakeholders including organizations or individuals who want to use the ability of machine learning usually do not have much technical knowledge or skillful recourses for software development [14, 15 & 16]. However, most of the machine learning tools are developed in the form of libraries for a specific programming language [e.g. 6, 38 & 39]. In order to embed machine learning modules into applications and utilize its functionalities properly, effectively, efficiently and robustly, senior programming abilities and considerable implementation effort are required, which are too costly for stakeholders who have very limited resources, not to mention that they still have to label the samples before they can actually make use of the applications. If they want to reduce the labeling
effort, they need to have even more technical knowledge and implementation effort.

1.4 Existing solutions and their limitations

1.4.1 Dedicated applications
Much research has been carried out in order to free various kinds of users from heavy development works and data analysis operations in different ways. Goble and Roure [9] introduced a workflow tool to optimize the operation on data-centric research involving numerous complicated procedures and huge computing power. An application was developed by the Social Media Research Foundation [10] to provide scientists a utility program for data analysis and relationship network visualization in order to speed up their work. For research in biology, Microsoft Research's eScience group [11] released the Microsoft Computational Biology Tools which contain 13 applications to serve 13 different biological exploration purposes mostly related to Human Leukocyte Antigen (HLA). For example, a statistical tool named PhyloD was provided in the package specifically designed to identify certain mutations and help researchers to decode the mutation rules.

However, these works were either introducing tools for a particular project and purpose, or approaches that are only suitable for extremely large-scale computation such as data-centric level analysis. As a result, only a small groups of users can benefit from these technologies. For stakeholders of small-scale machine learning with limited resources, they still have to develop, or ask professionals to develop a dedicated application to serve their own requirements. In addition, the aforesaid solutions did not address the problem of the heavy labeling tasks required of users of machine learning tools.

1.4.2 Machine learning application development frameworks
There were several projects carried out to provide some frameworks to reduce the implementation efforts as well as operation requirements required in
developing and executing machine learning applications. Low et al. designed a machine learning and data mining framework with distributed GraphLab and enabling the use of cloud resources [40]. It provided the ability of deploying machine learning applications at low cost cloud services such as Amazon EC2, and thus reducing the overall cost [40]. However, although this framework had solved the problem of getting computing resources and saving a part of the implementation effort, it did not address the problem of manual labelling tasks. Moreover, the configuration and deployment of the framework are complicated and unnecessary for small-scale machine learning.

Another example is the Big Dig for Scientific Data (BigDigSci) [41] framework, which was developed by Dr. Sarana Nutanong from City University of Hong Kong, Dr. Yanif Ahmad and Dr. Tom Woolf from Johns Hopkins University. It is a scientific exploration framework for general scientific purposes that made use of scheduling and active learning to lighten the manual effort from scientists during data exploration and management. It supported scientific data exploration applications, especially for those which require powerful computing resources for performing expensive simulation to determine the properties of samples.

![Figure 1. Differences that BigDigSci makes [9]](image)

Although the framework has flexibly solved the problem of lacking computing resources and high implementation effort, the framework does not address the problem of manual sample labeling. Also, the framework relies on command line
interpreter and does not have a graphical user interface. The use of command line interpreter requires extra knowledge and it is not convenient for users to use, adjust and monitor the framework since its input and output are limited to plaintext. More importantly, the models that machine learning builds consist of complicated structures and procedures which cannot be transformed to human readable textual representation. There is a need to implement a visualization method to provide a graphical representation for the generated model so the users can monitor, evaluate and review their learning results.

1.4.3 Integration of crowdsourcing and supervised learning
Crowdsourcing is an approach that allows requestors to obtain scalable workforce and services from a large crowd of people. A lot of research has been carried out to evaluate the feasibility and effectiveness of using crowd-sourcing approach to gather labels for machine learning and obtain positive results. For example, Callison-Burch et al. [42] gathered speech and language samples via Amazon Mechanical Turk (MTurk); Laws et al. [10] carried out experiments to explore the use of MTurk as an external annotation system for active learning; Ambati et al. developed an application that integrated MTurk services with active learning to build statistical machine translation models [43].

Although crowdsourcing services like MTurk were proven to be very effective and cost-efficient for supervised learning, especially the active learning approach, there were only a few real applications using crowdsourcing to solve the labeling problem. One of the main reason is that there is considerable implementation effort required to integrate crowdsourcing with machine learning. All the aforesaid integration attempts were achieved by developing a tailor-made application. So far no integration of crowdsourcing and active learning can serve general data analysis purposes.
1.5 Objectives and Scope

This project aims to develop a new machine learning framework integrating machine learning with crowdsourcing in order to offer a comprehensive solution to the sample labeling problem, study time problem and implementation effort problem so that organizations or individuals with limited skills and resources can also benefit from machine learning. The solution makes use of scikit-learn [6] (a well-known open source machine learning library written in Python) for providing machine learning ability, Amazon Mechanical Turk [9] (a popular crowdsourcing platform) for providing crowdsourcing ability, and Flask [44] (a lightweight web application framework written in Python) for providing graphical user interface.

In order to solve the sample labeling problem, the framework has the ability of using and managing Amazon Mechanical Turk services to automate the active learning process so that users can be freed from annoying labeling tasks as well as speeding up the learning process with the advantages of crowdsourcing. For the study time problem and implementation effort problem, this framework generalizes most of the sampling, labeling and learning procedures so users only need to implement a few lines of programming code while the rest of development can be finished in just a few clicks via the web-based graphic user interface. The framework also provides comprehensive performance evaluation tools for users to monitor the learning progress and evaluate the results. These greatly reduces the technical knowledge and study time required.

Although the framework generalizes most of the machine learning procedures, the framework maintains a great level of flexibility on active learning strategy. Users can configure the framework to utilize almost all the machine learning algorithms and settings that scikit-learn supports. Users can even extend the framework with customized machine learning kernels that scikit-learn does not provide. In addition, the framework is implemented as a web application supported by Flask so users can access the framework like browsing a website. In
other words, if the framework is hosted on a server, it can be remotely accessed as a cloud services.

2. Literature Review

2.1 Machine learning categories

2.1.1 Unsupervised learning

Machine learning uses algorithms to learn from data while unsupervised learning is a branch of machine learning that algorithms learn from unlabeled data. A lot of research has proven that unsupervised learning is useful for summarizing some given data, identifying their key features and doing clustering and dimensionality reduction [45 & 46]. However, since all the samples offered to learners are unlabeled, the solution built by unsupervised learning cannot provide any error signal or positive feedback for the evaluation of correctness. Moreover, although the pattern recognition ability provided by unsupervised learning can be used to reduce the dimensionality of datasets, it is not necessarily required in scientific research environment which puts priority on exploring how the data can be used to benefit the society rather than what the data are. Similarly, commercial applications of machine learning focus on making prediction which unsupervised learning can hardly fulfill. Experiments were carried out by Le et al. [47] to explore the ability of unsupervised learning in building high-level classifiers and they proved that it is possible to obtain class-specific detectors with only unlabeled samples. However, the experiment took 10 million samples with 1 billion connections and executed on 1000 machines consisted of 16,000 cores to get a sensitive model. It is unworthy to be adopted in real-world practice.

2.1.2 Supervised, semi-supervised and active learning

Supervised learning is another machine learning discipline. Unlike unsupervised learning which learns from unlabeled data, the learner of supervised learning
learns from those labeled. The concept is similar to a learner being trained by a teacher by showing him some examples. Semi-supervised learning is similar to supervised learning, but the training dataset given to the learner is incomplete in a way that most of their labels are missing. Active learning is one of the semi-supervised learning approaches in which the learning process can evaluate the value of an unlabeled sample and has control on which samples should be included in the training. This approach allows the learner to build a more accurate prediction model with fewer samples and labeling tasks required, as well as accelerating the learning process [16, 17 &18]. It is great for situation in which data can be collected or generated in a low cost while sample labeling tasks are expensive. It matches with the case of scientific exploration and commercial environment which abundant data can be collected with instruments or system logs, but resources for labeling are very limited [13, 14 & 16]. Moreover, the nature of active learning is consistent with many general scientific research and commercial application purposes, which are enhancement on phenomenon identification and prediction. That is why active learning is widely adopted in many existing machine learning applications (e.g. [17, 18, 19, 20, 21 & 22]). However, even though active learning selects only the most desirable samples and thereby reduces the annotation effort from users, building a scientifically accurate classification or prediction model may still need thousands of samples or even more to be labelled. More importantly, the time required for labeling a sample could be much longer than the generation of data. The case happened to Zhu [23] during the training of a model for speech recognition is an authentic example which the annotation of audio samples could take 10 to 400 times longer than the actual speech depending on the precision required. In light of this, more effort should be put to improve the active learning procedures and reduce the manual effort required from users.
2.1.3 Reinforcement learning

Reinforcement learning is another machine learning area that is concerned with how a machine can learn to take action or make reaction in a sequential dynamic environment to maximize long-term reward for a particular objective. Unlike supervised learning, the training data from the environment cannot be used to evaluate and signal its correctness [48 & 49]. It is usually discussed together with game theory and control theory since it is widely applied for adaptive optimal control of nonlinear systems including robotics, games and some control automation systems [50]. For example, Nate and Peter [51] used reinforcement learning to train a quadrupedal robot to learn gait for fast locomotion; Bagnell [52] applied reinforcement learning with policy search methods to allow autonomous helicopter control; Tesauro [53] trained a computer backgammon program with temporal difference learning algorithm and achieved a level of playing backgammon very close to the top human players at that time.

Without doubt, reinforcement learning is a very useful technical in artificial intelligence control. However, the usage of reinforcement learning is greatly distinct from general machine learning goals. That why reinforcement learning are usually developed in independent libraries but not contained in most of the popular machine learning libraries including scikit-learn.

Summary
In order to maximize the usage and applicability of the framework so that it can best serve the machine learning goals of general users, active learning was selected to be supported in this project.

2.2 Machine learning output models

2.2.1 Cluster analysis
Cluster analysis is concerned in grouping objects in order to maximize the similarity between objects within the same group and minimize the similarity
between objects from different groups. It is usually considered as an unsupervised learning problem, yet it can be achieved in a semi-supervised learning approach [34, 54]. One of the most popular uses of cluster analysis in recent years is social networks. Cluster analysis is used to recognize communities or a group of people who share similar interests or background to help users to make new friends [55]. Cluster analysis can also be a solution of recommender systems [56]. By clustering users with similar interest together, there is a promising chance that a user in a cluster is interested in the products that another user in the same cluster has brought. As a results, it can help sellers to deliver their advertisements to high potential customers. Yet, cluster analysis is usually not the unique solution to the goals it serves. Especially in many cases clustering problems can be transferred into classification problems with proper assumptions. (Please see 2.2.3 Statistical classification)

2.2.2 Regression analysis
Regression analysis is concerned in making estimation on whether relationships exist between variables as well as what is the strengths and directions of the relationships. It is usually considered as a problem under the discipline of supervised learning [34]. In other words, regression analysis can help oneself understand the dependency of variables, and thus making prediction on the effect caused by the changes of variables. For example, Krasnopolsky and Fox-Rabinovitz [2] applied this approach to climatology and created prediction models for climate modeling and weather prediction. Some insurance company perform regression analysis on customers’ personal data such as background, lifestyle, driving experience and past accidents to estimate the amount of insurance premium [57]. Similar approach can also be adopted in forecasting stock market.

There are many regression analysis methods developed to suit different scenarios and requirements, such as linear regression for simple linear tendency and logistic regression for binary response prediction. A complicated example could
be isotonic regression, which tries to find the best weighted least-squares that fit a set of data represented in relative positioning [58]. The performance and accuracy of prediction made by regression analysis greatly depends on the correct selection on regression approaches and the data generated for analysis. Users must have advanced knowledge in regression analysis to make correct decision. Otherwise, the regression methods may give misleading results and lead to unpredictable loss.

2.2.3 Statistical classification

Classification is a machine learning and statistics problem that is concerned with which of the categories does a new input belongs to. It is usually considered as an instance of supervised or semi-supervised learning in which classifiers are trained with samples which their categories are known already [34]. The usage of statistical classification is wide and most of the application examples mentioned in the introduction can be considered as classification problems. For example, spam filtering can be achieved by training a classifier with existing emails labeled as either “spam” or “non-spam” to classify new incoming emails. In scientific research, the objective of doing classification is consistent with many research purposes, including classifying the type of virus, disease, DNA sequence, chemical composition and chemical structure. [19, 20, 21, 22, 23 & 24]

In addition, statistical classification can be the work-around of other machine learning problems, including cluster analysis and regression analysis. The clustering problem, which is related to grouping similar samples into clusters, can be transformed to a classification problem by considering each cluster as a category. With this definition, the action of classifying samples to different categories actually has the same meaning as assigning samples to different clusters. That is also the reason for classification’s being often treated as the supervised learning version of unsupervised clustering [59]. For regression problem, which is concerned on making prediction based on historical data, we can consider historical data as samples and simplify the known consequence to
be corresponding labels. Then the meaning of classification will become exploring what consequence the new data may lead to. For example, if we want to have a model that can predict whether a typhoon will happen, we can train a classifier with weather conditions in some moments in the past as samples, and whether a typhoon really appeared after that time (true or false) as labels. Then the trained classifier will be able to classifier or deduce the likelihood of whether the current weather conditions can lead to the appearance of a typhoon. That is why most of the algorithm for classification problems, such as Support Vector Machine and Decision Trees, can also be extended to solve regression problems [60 & 61].

**Summary**

*Since the available development period for this framework is limited, it is unrealistic to make the framework to support all three kinds of machine learning problem. Since the classification problem has the best applicability and potential to be the workaround of the other two problems, the framework mainly focuses on optimizing of use of classification algorithms.*

**2.3 Sampling methods**

**2.3.1 Stream-based selective sampling**

In stream-based selective sampling for active learning, samples are collected or generated only when samples are being queried by the learner. After the samples are collected, they are passed to the learning process which makes judgments on whether a sample is desirable to be labelled. Figure 2 provides an illustration on how stream-based selective sampling works.
Figure 2. Illustration of stream-based selective sampling operation

This approach is direct and easy to implement, and it does not require much effort on sample manipulation such as sample sorting and searching [62]. However, it is only practical when samples can be obtained immediately at very low cost [8]. For scientific exploration, although data can be gathered at low cost, it is hard for instruments to meet the query speed because of the startup and preparation time. Similarly, in a commercial environment, samples used for learning are usually historical data and it is impractical for the environment to generate suitable data on the fly. Also, as mentioned by Atlas et al. [63], the unknown distribution of samples can greatly affect the efficiency of stream-based sampling. If samples are unevenly distributed, the chance of a sample meeting the threshold of query criterion can be very low. In other words, huge amount of samples may need to be collected and judged by the learning process until a desirable one appears (In Figure 2, It is represented by the loop in the top-right corner). Since the distribution of data collected with instruments or environments is not guaranteed, this limitation adds uncertainties to the efficiency and robustness of the learning process.

2.3.2 Pool-based sampling

In pool-based sampling for active learning, a pool of unlabeled data is pre-generated and stored for sampling. Unlike stream-based selective sampling in which data are gathered sequentially and decisions are made individually, the
pool-based approach gathers data in onetime and selects samples from the entire collection of ranked data [8]. Figure 3 provides an illustration on how pool-based sampling works.

**Figure 3. Illustration of pool-based sampling operation**

For scientific research in which mostly data come from instruments, one time collection has better applicability than sequential gathering. Similarly for commercial environment, historical data can be easily accumulated. Although pool-based sampling requires ranking the whole pool of data, many studies have been done and proven its superiority. For example, Lewis and Catlett [64] examined uncertainty sampling with a cheap classifier for ranking to train an expensive one in pool-based environment and proved its advanced effectiveness and efficiency. Research done by McCallum and Nigam [65] also confirmed that pool-based sampling can give better accuracy than other metrics.

This sampling method is already adopted in many applications to solve real-world problems, such as text, image, speech and video identification and retrieval. It is also utilized in scientific research. For example, Lapedes [1] used active learning with pool-based sampling method to perform DNA analysis; Krasnopolsky and Fox-Rabinovitz [2] applied this approach in climatology and created prediction models for climate modeling and weather prediction; Liu [66] also used a similar approach to train models for cancer diagnosis.
Summary
Since pool-based sampling is more flexible and practical than stream-based selective sampling, it is chosen to be used in this project.

2.4 Amazon Mechanical Turk (MTurk)
Amazon Mechanical Turk (MTurk) [9] is a popular crowdsourcing platform for companies and organizations to solve problems that involve large amount of tasks requiring human Intelligence. By creating human Intelligence tasks (HITs) on the Amazon platform and opening for answers, requesters can obtain scalable and on-demand human workforce at low cost.

2.4.1 Applicability
Currently on the Amazon Mechanical Turk platform, one of the most popular type of HITs is created by small and medium-sized companies that ask workers to transform some images of old receipts into text and number. It is indeed very similar to labeling tasks in supervised learning, especially the classification problem. Moreover, with advanced setting, it is possible to display comprehensive multimedia information on MTurk and ask for various kind of answers. For example, Sorokin et al. [67] implemented a module in Java applet and asked workers to trace the boundary of the person shown on a picture. Laws et al. [8] combined active learning and MTurk on their application and discovered that MTurk could considerably reduce the cost of annotation and improve the learning performance. All these show that Amazon Mechanical Turk is suitable to be used as the annotator in learning process. In fact, Amazon Inc. itself is using this platform for testing and improving their online shopping recommender system by asking workers to make judgment on whether a product is a justified recommendation for another (as shown in Figure 4).
2.4.2 Accuracy, efficiency and cost

MTurk allows requestors to setup qualification requirements of workers such as expertise, amount of HITs finished and the radio of approved HITs. At certain level it helps requestors to ensure the high quality of labels. However, on the MTurk platform, requesters only need to pay workers a small amount of money for each task they answer, and since everyone can register on MTurk and become a worker, the quality of answers gathered via MTurk is controversial. Yet, the experiments carried out by Kittur et al. [68] disproved this concern. In several experiments on the answering speed and quality of answers returned by MTurk workers, it is discovered that with appropriate task design and question partition, answers could be gathered rapidly with invalid response rate only 2.5% and around 70% of them were consistent with the answers given by professionals. Similar results were found by Paolacci et al. [69] who revealed the fact that the quality of answers returned from MTurk was as good quality as those gathered from a university. A more comprehensive exploration was made by Sorokin et al [67]. In their experiments, workers was asked to finish different complicated labeling on some pictures, such as identifying the joints of human body and
drawing the boundary of a person (as shown in Figure 5). The results showed that most of the workers tried to accomplish the task and a very high percentage of them did give qualified answers while most of the unqualified answers were caused by confusing samples, such as joints on the edge of an image, which could be solved by giving better instructions to workers.

![Instructions displayed on MTurk](image)

**Instructions displayed on MTurk**

- This is an experimental project. The makers of the app may change later.
- Instructions:
  - Block on the joint
  - Label: person
  - Add shape: person

**Unqualified answers**

- ![Unqualified answers](image)

**Qualified answers**

![Qualified answers](image)

**Figure 5. Instructions and some answers of the experiment that asked workers to label the joints' position.**

It is convincing that the labels gathered via MTurk are qualified to be used in supervised learning. Also, the efficiency of label gathering is promising. Benefited from 500,000 of registered workers, tasks usually can be finished rapidly. In the aforesaid joints positioning example, labeling 915 images was finished within 2.5 hours [64]. Ipeirotis et al. [70] also conducted an experiment on MTurk and discovered that given a task of classifying websites into one of four categories
according to the degree of containing adult contents, a trained intern paid with $15 per hour could complete 250 websites per hour while MTurk could yield 2500 per hour at a lower overall cost.

2.5 Summary of literature review
This project aims to provide an easy to use machine learning framework that can lighten the implementation effort of machine learning applications, free users from heavy labeling tasks and thus encourage the use of machine learning in all scale for general scientific and commercial purposes. With this goal in mind, after comparing the nature and applicability of different machine learning approaches and problems, supervised learning focusing on classification problem is selected. Meanwhile, pool-based sampling is used to adapt the framework to suit various scientific and commercial environments. Moreover, Amazon Mechanical Turk is used to achieve automation of active learning in order to solve the labeling problem.
3. Proposed design, solution, system

3.1 Solution overview

The design concept of the solution can be visualized as follows:

![Diagram of the framework]

**Figure 6. Design of the framework**

In Figure 6, the frames in white color are existing tools or applications used or called by the framework while the frames filled with blue color are items that are developed in this project. The frame in yellow represents the part which requires some configuration or implementation from users.
3.2 Execution environment and programming language

Table 1. Execution environment and packages used

<table>
<thead>
<tr>
<th>Fields</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>Server side: Linux/Ubuntu</td>
</tr>
<tr>
<td></td>
<td>Client side: Any</td>
</tr>
<tr>
<td>Programming languages</td>
<td>Python 2.7, Jinja2, HTML, CSS and JQuery</td>
</tr>
<tr>
<td>Supported Browse</td>
<td>Chrome and Firefox</td>
</tr>
<tr>
<td>Database</td>
<td>PostgreSQL</td>
</tr>
<tr>
<td>Libraries and external</td>
<td>Server side:</td>
</tr>
<tr>
<td>resources used</td>
<td>1. Scikit-learn (machine learning library)</td>
</tr>
<tr>
<td></td>
<td>2. Flask (web application framework)</td>
</tr>
<tr>
<td></td>
<td>3. Psycopg2 (PostgreSQL adapter)</td>
</tr>
<tr>
<td></td>
<td>4. Numpy (array processing library)</td>
</tr>
<tr>
<td></td>
<td>5. Amazon Mechanical Turk Command Line Tools</td>
</tr>
<tr>
<td></td>
<td>6. Java Runtime Environment (JRE)</td>
</tr>
<tr>
<td></td>
<td>Client side:</td>
</tr>
<tr>
<td></td>
<td>1. Bootstrap (CSS style)</td>
</tr>
<tr>
<td></td>
<td>2. Bootstrap-select (Supportive CSS style)</td>
</tr>
<tr>
<td></td>
<td>3. Ace (code editor)</td>
</tr>
<tr>
<td></td>
<td>4. D3 (SVG rendering support)</td>
</tr>
</tbody>
</table>

3.3 Component explanation

3.3.1 MTurk Connector

MTurk connector is developed in the project in order to provide Python applications the ability of interacting with MTurk services with the use of Amazon Mechanical Turk Command Line Tools (MTurk_CLT), which is a command line application developed by Amazon using JAVA that allows users to communicate with MTurk services using commands under UNIX based system.
The **MTurk connector** layer parses user inputs into suitable format that **MTurk_CLT** required and calls its commands to perform operations including checking MTurk account balance, creating HITs, checking HIT status, approving HIT submissions, retrieving answers, and removing HITs. **MTurk connector** also parses, manages and stores the output from **MTurk_CLT** (responses from MTurk server) into suitable format for the uses of upper layers. It also has the ability of detecting errors in the user inputs such that HITs creation or approval can be suspended beforehand and no money will be wasted on invalid HITs.

In addition, this class can be executed as long as **MTurk_CLT** and **PostgreSQL** are in place. In other words, the usage of this class is not limited to active learning. It can be used for any purposes that require Amazon Mechanical Turk service as a crowdsourcing platform.

### 3.3.2 Learning automation layer

**Learning automation layer** contains the implementation for performing automated machine learning. It accepts configuration details of machine learning from upper layers and then coordinate with **scikit-learn** and **MTurk connector** to perform automated active-learning. It also has several sampling strategies, end-of-learning strategies, error-validation methods and label buffering implemented and offered to user to select.

In order to achieve auto-labeling and thus automatizing the active learning operation, this layer is also responsible for managing MTurk account in high level means. In each iteration of sampling, unlabeled samples selected by the sampling strategy will be passed to **MTurk connector** for HITs creation. This layer frequently queries **MTurk connector** to track the status of HITs. Once **MTurk connector** returns a positive feedback, this layer accesses the database to retrieve the answers, parses them into the format of labels and passes to the learner for the next iteration of learning.
3.3.3 Web application (server-side)

Users can use this framework as a web application using major browsers while the server side of the web application is supported by Flask. This layer is responsible for accepting, parsing, validating and verifying user input, then calling low layer functions according user requests, and responding to user with suitable HTML page rendering or redirecting along with feedback messages given by this or lower layers. Simply speaking, it is the middle man of the graphical user interface and the lower layers which perform machine learning operations.

Except being an adaptor, this layer also implemented the concept of “project”. It allows the existence of multiple projects, each of which has its own machine learning application. Management of projects is accomplished mainly by this layer with calling some supporting functions provided by lower layers.

3.3.4 Graphical user interface

The Graphical user interface of the framework is in a form of web application and it can be accessed through major browsers. The HTML pages and components are rendered by Flask at runtime with JQuery supports on client side. The graphical user interface allows users to configure and monitor the framework, projects, MTurk accounts, learning operations and samples.

3.3.5 MTurk Client application

A client application is the machine learning application created by the framework according to user requirements. After configuring a project with proper scikit-learn and MTurk setting, the framework will allow users to generate an application. The application can be executed either through the framework or independently outside the framework as long as all necessary resources are in place.

Although the framework generates client applications automatically according to user setting, two functions are necessary to be implemented by users
themselves. One is the function defining how samples are transform into information to be displayed on MTurk. Another one defines how answers returned by MTurk are transformed into labels. Each function usually requires just a few lines of coding. Users can implement these two functions on the code editor provided by the graphical user interface.

4. Detailed methodology and implementation

Since the project will be continued by future students, some of following contents are explained in detail regarding the structure and coding from server side to interface implementation in order to help them to understand the operation of the framework quickly.

4.1 Communication with MTurk

4.1.1 Database design for MTurk

4.1.1.1 MTurk Pending table

The pending table stores the details of human intelligence tasks (HITs) that have been created on the Amazon server but not yet answered by workers, or the answer is not yet retrieved. In other words, the automated process is pending for the answers of these HITs. Each project has its own pending table named as <projectName>_pending. The detailed table structure is as follows:

Table 2. The MTurk pending table design

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success_id</td>
<td>serial</td>
<td>Serial id for each successful upload</td>
</tr>
<tr>
<td>raw_data</td>
<td>text</td>
<td>Raw data (samples) object in text</td>
</tr>
<tr>
<td>data</td>
<td>text</td>
<td>Transformed data that were uploaded to MTurk</td>
</tr>
<tr>
<td>hitid</td>
<td>varchar(30)</td>
<td>HIT ID generated by MTurk</td>
</tr>
<tr>
<td>hittypeid</td>
<td>varchar(30)</td>
<td>HIT type ID generated by MTurk</td>
</tr>
</tbody>
</table>
When the automated learning process wants to check for HITs’ status, it will query the Amazon server according to information stored in this table. Once an answer is retrieved or expiration of a HIT is detected, the corresponding record will be removed from this table so no redundant results and effort are made.

4.1.1.2 MTurk Record table

The record table stores information of all HITs that their life cycle are ended normally (either answered or expired). Each project has its own record table named as `<projectName>_record`. The detail table structure is as follows:

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>success_id</td>
<td>serial</td>
<td>Serial id for each successful upload</td>
</tr>
<tr>
<td>raw_data</td>
<td>text</td>
<td>Raw data (samples) object in text</td>
</tr>
<tr>
<td>data</td>
<td>text</td>
<td>Transformed data that were uploaded to MTurk</td>
</tr>
<tr>
<td>hitid</td>
<td>varchar(30)</td>
<td>HIT ID generated by MTurk</td>
</tr>
<tr>
<td>hittypeid</td>
<td>varchar(30)</td>
<td>HIT type ID generated by MTurk</td>
</tr>
<tr>
<td>reward</td>
<td>float</td>
<td>Reward of this HIT</td>
</tr>
<tr>
<td>creation_time</td>
<td>timestamp</td>
<td>Creation time of the HIT</td>
</tr>
<tr>
<td>answer</td>
<td>text</td>
<td>Answer that retrieved from MTurk</td>
</tr>
<tr>
<td>Accept_time</td>
<td>timestamp</td>
<td>Time that the HIT is accepted by a worker</td>
</tr>
<tr>
<td>submit_time</td>
<td>timestamp</td>
<td>Submission time of the answer</td>
</tr>
<tr>
<td>is_expired</td>
<td>boolean</td>
<td>Whether the HIT is ended with answer or expiration</td>
</tr>
</tbody>
</table>

This table is created for recording and source tracing purpose. Users can use the records in this table to trace all the labels gathered from MTurk. Its uses includes label quality evaluation, payment tracing, performance analysis, etc.
4.1.1.3 MTurk Failure table

The failure table stores information of HITs that fail to be created. In other words, the contents in this table are not submitted to Amazon Mechanical Turk server. Each project has its own failure table named as <projectName>_failure. The detail table structure is as follows:

Table 4. The MTurk failure table design

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>failure_id</td>
<td>serial</td>
<td>Serial id for each HIT creation failure</td>
</tr>
<tr>
<td>raw_data</td>
<td>text</td>
<td>Raw data (samples) object in text</td>
</tr>
<tr>
<td>data</td>
<td>text</td>
<td>Transformed data that are planned to be uploaded to MTurk</td>
</tr>
<tr>
<td>message</td>
<td>text</td>
<td>Error message that MTurk_CLT returns</td>
</tr>
</tbody>
</table>

Users can use the information stored in this table to find out why some samples are invalid for Amazon Mechanical Turk and then make correction.

4.1.1.4 MTurk Answer table

The answer table is created to store the mapping of samples and answers while HIT ID indicates the source of labels. These sample-answer pairs represent those samples in which the answers are already retrieved from the MTurk server but not yet used by the learning process. Each project has its own answer table named as <projectName>_answer. The detail table structure is as follows:

Table 5. The MTurk answer table design

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>raw_data</td>
<td>text</td>
<td>Raw data (samples or sample identifier such as sample id) in text</td>
</tr>
<tr>
<td>answer</td>
<td>text</td>
<td>Answer that retrieved from MTurk</td>
</tr>
<tr>
<td>hidid</td>
<td>text</td>
<td>ID of the HIT that the answer come from</td>
</tr>
</tbody>
</table>
In other words, it is where the learning process get new training data from. Once the learning process uses these pairs as training data, they will be transferred to the labeled sample pool and removed from this table.

### 4.1.2 MTurk Connection

Connection to the MTurk server is done by calling MTurk command line tools. In order to connect to MTurk, users must provide two value: AWS Key and AWS Secret Key. AWS Key is a fix length string (20 characters) used to uniquely identify each MTurk account while AWS Secret Key is another fix length string (40 characters) that is for verification. The command line tool gets these two values by reading a file called `mturk.properties` which is supposed to be stored under the `bin` directory along with the shell scripts (.sh) files. However, the command line tool has an implementation fault in which its scripts are not looking for `mturk.properties` under `bin`, but in the current directory of the command line interpreter. Since the framework is not under the `bin` directory, calling commands will cause a ‘file not found’ error. In order to solve this problem, the framework accepts AWS Key and AWS Secret Key as parameters and create a temporary `mturk.properties` file under the current directory so that the command line tools can find it when it is called by the framework.

Once the `mturk.properties` is in place, access to MTurk server can be done by calling commands. A function `def run_cmd(cmd)` is implemented to run external commands using the `subprocess` library. For example, to test whether a connection to the MTurk server can be established, the framework tries to execute `run_cmd("bin/getBalance.sh")` to get the balance of the MTurk account. At the end of command execution, the function returns the command output. In case that users execute the learning applications via command line instead of the framework’s GUI, the `run_cmd` function will read the standard output stream and yield output instantly on the command line interpreter instead of returning output at the end of execution.
In addition, MTurk provides a sandbox environment for developers to test HITs creation or other activities. Thus, the framework allows user to user the sandbox as well. Users can set whether they want the execution to be carried out in the sandbox or the real platform, then Mturk_connector will pick up users’ selection and add –sandbox argument to all the command.

4.1.3 HIT creation

HIT creation is achieved by the loadHITs command. This command has four necessary arguments: properties, question, input and label. The label argument specifies the output name of the batch of HITs to be created while the properties, question and input argument specify the path of .properties, .question and .input file (see 4.1.3.1 Properties file, 4.1.3.2 Question file and 4.1.3.3 Input file). Also, after creation, the command line tool will generate a success file named <label>.success and a failure file named <label>.failure file. The success file contains the HIT_ID of each successfully created HIT while the failure file contains information of failed creation.

4.1.3.1 Properties file

HIT creation requires a properties file that specifies the HITs’ properties including title, description, keywords, annotation, reward, duration, lifetime and auto approval delay. These properties must be in proper format and stored in a file with file extension .properties. In order to free users from creating this file manually, the framework provides a page for users to input these value as shown in Figure 7.
The form restricts the value type of different properties and provides default value for some fields, such as reward and duration. The framework will allow users to upload a properties file instead of filling the form.

After receiving user setting or a properties file, the file is stored under the project directory and named as `mturk.properties`. The framework has a template of properties file with placeholder for each property. If users choose to build a new properties file with the form, after receiving user inputs, values in the request will be parsed into proper format (e.g. int, float or string) and rendered using the template by the web application (server-side) layer with `.format()` function. Since the `.format()` function consider `{` and `}` as the start and end notation of
placeholders, the framework will check whether the user inputs contain literal ‘{’ and ‘}’, and replaces them with ‘{{’ and ‘}}’ for escape purpose.

After configuring the properties file, each project that enables MTurk labeling ability must have an `mturk.properties` file placed under its directory. Thus when the `loadHITs` command is called, it will have the following argument and value:

```bash
> loadHITs.sh ... -properties '<projectName>/mturk.properties' ...
```

### 4.1.3.2 Question file

HIT creation requires a question file (with extension name `.question`). It specifies the contents that is going to be displayed on MTurk for workers to view and answer. This file uses XML with structure much more complicated than the properties file. For example, a simple question file for creating a HIT consisting of 10 questions with selection answers must have at least 250 lines of XML syntax. Moreover, its contents must be in XML format that is consistent with the XML Schema Definition (XSD) stored remotely on Amazon server:

http://mechanicalturk.amazonaws.com/AWSMechanicalTurkDataSchemas/2005-10-01/QuestionForm.xsd

In order to free users from setting up the question file, the framework allows users to upload an existing question file or build a new question file using the interface by filling a form like Figure 9.
Since each question represents the labeling task of a sample, the framework allows users to create question file that a question is repeated multiple times for multiple samples. So the minimal reward of a HIT, which is USD 0.01, can actually be paid for more than labeling one sample, thus increasing the flexibility and reducing the cost. Besides the number of question, the form also requires users to input the question title, question overview, display name and question text content. Similar to the properties file, web application (server-side) layer will do the escape of ‘{’ and ‘}’.

Moreover, users must select the type of answer that they expect to get from workers. The framework supports all four types of answers that MTurk supports, including free text answer, free text answer with regex, numeric answer, and selection answer. With the support of JQuery, when free text answer with regex is selected, two more fields will be displayed to accept the regex (regular expression that JavaScript uses to validate the input format) and an error message which is displayed to workers when the regex validation fails (as shown
in Figure 10). Similarly, when numeric answer is chosen, two fields will be displayed for users to configure the maximum and minimal value of the available numeric input.

![Figure 10. Fields for inputting regex and error message field for Free text answer with regex](image)

For selection answer, when it is selected by users, an extra block will be displayed for user to add new selections (as shown in Figure 11).

![Figure 11. An extra block for user to add new selection](image)

For each selection a user adds, a new field will be created for users to define the value that the new selection represents. For example, users may add two selections with display text True and False, while True represents value 1 and False represents value 0:

![Figure 12. A selection True with value 1 is created. Another new selection named False is going to be added](image)
Since the fields for inputting selection value are created dynamically using JQuery, in order to let the web application (server-side) layer be able to retrieve their value, each new input fields is generated with name sel_<number>, where <number> is counted by JavaScript starting from 0. Also, for each selection created, a hidden input field will be generated to store its selection display text. Each of these hidden input fields has name seln_<number>. On the server side, it retrieves these values by checking the existence of form.request['sel_<number>'] and form.request['seln_<number>'] starting from <number> equal to 0 until the statement return False.

After retrieving user setting from the form, the framework renders a question file using netted rendering strategy. The framework has a question file template with placeholders for question title, question overview, and questions. There are four question templates for four types of answers while each of them contains placeholders for question identifier, display name and question content. The free text answer template further contains placeholders for regex and error message; the numeric answer template contains placeholders for maximum and minimum value; and the selection answer template contains an extra placeholder for selections. Finally, there is a selection template that has two placeholders for the name and value of a selection. During the rendering process, the web application (server-side) layer picks templates to be rendered recursively according to user selections. For example, if a user requires 5 questions in a HIT with 2 selections for each question, the server will first open the selection template and render its XML representation twice with .format(); then open the selection answer template and render it with question identifier, display name, and the pre-rendered selections; finally open the question file template and render it with question title, question overview, and 5 times of the pre-rendered questions.

Also, in view that plain text contents usually cannot express rich enough information, the framework allows users to input HTML contents such as <img
in the question overview field and question content field. When the question file is rendered, these fields will be enclosed with `CDATA` tab so that the HTML tab will not disturb the XML format required in the XSD. In addition, since each question refer to a labeling task, each question must have some contents related to a specific sample. These sample related contents can be specified with placeholder in format like `${image}` (for its detail usage, please see 4.1.3.3 Input file). Thus, the question contents will be rendered like the following:

```xml
<QuestionContent>
  <FormattedContent><![CDATA[<p>Is the following image a car?</p><img src="${image}" /></]]></FormattedContent>
</QuestionContent>
```

If users require the question to be repeated more than once in a HIT, the framework will automatically append a unique number to the end of the placeholder for each question (e.g. `${image1}` for question 1, `${image2}` for question 2 and so on) so each question has a unique placeholder.

After configuring or uploading a question file, each project that enables MTurk labeling ability must have an `mturk.question` file placed under its directory. And thus when the `loadHITs` command is called, it has the following argument and value:

```
> loadHITs.sh ... -question `'<projectName>/mturk.question'` ...
```

4.1.3.3 Input file

As mentioned in section 4.1.3.2 Question file, placeholders in format like `${image}` is used to display sample dependent contents in each question. The input file contains the concrete data used to replace these placeholders. It is a CSV file with tab `\t` as its column separator. Each row in the input file represents the content for a HIT while each column represents the data for a placeholder.
Therefore, the amount of columns in the input file should be equal to the amount of placeholders specified in the question file.

However, the framework only has the knowledge of samples and features while the users may want to display samples to workers in a different format on MTurk. For example, if the samples and features are RGB values of colors, users may want to display some color images on MTurk with `<img src="${image}">` specified in question file and thus the data in the input file to replace `${image}` should be URLs of images. To achieve this effect, users can implement a function `def sample2Info(self, sample, feature)` to transform a sample and its features to the data used to replace placeholders. It is an abstract function in a class named `LearningAutomation`. If users define one placeholder for each question, the `sample2Info` function should return a single value that used to replace that placeholder. If users define more than one placeholder for each question, the `sample2Info` function should return a list of values in which its length is the same as the amount of placeholders. In addition, taking into account that users may need variables maintained across functions, an empty `dict` object named `userDefine` is pre-created as an instance variable. For example, if users need a counter counting the amount of samples being uploaded in order to return a unique value for each sample, they can define it as `self.userDefine('count')`.

The graphical user interface has a Python code editor that allows users to implement this function with a default implementation provided. The code editor this framework used is the ACE editor, which is the same editor that Cloud9 IDE uses. The page for implementing this function is shown in Figure 13.
The editor is set with `maxLines = Infinity` and `tabSize = 2` to provide a better usability for Python. However, this editor is formed by some netted `div` blocks controlled by JavaScript but not an `input` or `textarea` field. In order to submit the codes with the form submit action, JavaScript is used to extract the value in the editor’s `div` blocks and store it in a hidden input field before submission.

Once the users have this function implemented, the framework can automatically generate input files for the unlabeled samples being selected by the active learning module. In addition, although the command line tool requires the input file to contain a header name for each column same as the corresponding placeholder name, the framework can detect placeholders in question files and append a header row to the front of the input file automatically so users do not need to bother with it. In other words, users only need to provide the implementation of `sample2Info(self, sample, feature)` once, then the framework can handle HITs creation for all samples automatically.
4.1.3.4 Success file

For the execution of `loadHITs` command, the MTurk server responses with a list of `HIT_IDs` and they are stored in a file with extension `.success`. Other operations of the command line tool, such as retrieving result and approving HITs, rely on this file to identify which HITs should be involved. Since the `.success` file is created and overwritten by the command line tool in each `loadHITs` operation, the `.success` contains only the `HIT_IDs` of the HITs created in latest `loadHITs` operation. However, the framework allows users to configure that new HITs should be created when the amount of unanswered HITs on the MTurk server is lower than a specified value. In other words, there may be some HITs pending for answers or retrieval but are not recorded in the `.success` file generated by the command line tool. In order to solve this problem, several tables are designed to store the information including `HIT_IDs` of all HITs (as mentioned in 4.1.1 Database design related to MTurk). For each operation that requires `.success` file, the MTurk connector first queries the database and then generates a new `.success` file containing all pending HITs to overwrite the existing one. Thus, the one created by command line tool is actually not used by the framework.

4.1.4 HIT management

4.1.4.1 Checking HIT status

In each iteration of learning, the active learning module selects unlabeled samples and uploads them to MTurk for workers to do labeling. After creating HITs, the learning automation layer periodically checks whether some HITs on MTurk are answered and are ready to be retrieved. The interval of checking is specified by users and the checking operation is accomplished by MTurk connector and MTurk command line tool with command `getResults`. The `getResults` command has two arguments: `successfile` and `outputfile`. The argument `successfile` is the path to the success file, which is a file that contain a list of `HIT_IDs` as mentioned in section 4.1.3.4 Success file. The `getResults`
command queries the MTurk server and retrieves the information of the HITs specified in the success file. A success file named ‘mturk.success’ is generated by MTurk connector at runtime with a list of HIT IDs retrieved from MTurk pending table which stored information of HITs that are not yet expired or answered (section 4.1.1.1 MTurk Pending table). After the execution of getResults command, the information responses from MTurk server will be stored in a file with a name specified by outputFile argument, which is ‘mturk.results’ as specified in MTurk connector in default. The mturk.results file contains information including HIT ID, HIT type ID, HIT properties, HIT status, Work ID, assignment accept time, assignment submission time and answers. MTurk connector reads and parses this file to check the HIT status and determine whether a HIT is expired, answered, or still waiting for answers. If a HIT is expired, the corresponding row in the MTurk pending table will be removed and samples will be returned to the unlabeled sample pool. If a HIT is answered, its row will be moved from MTurk pending table to MTurk record table and rows containing samples and answers will be inserted into the MTurk answer table waiting to be picked up by the learning automation layer.

4.1.4.2 Retrieving answers

After the execution of getResults command, sample-answer pairs are stored in MTurk answer table. When the MTurk connector checks for HITs status and return True indicting that some HITs were answered, the learning automation layer will access the MTurk answer table to get new labels. At this moment, the answer format is the same as the one specified in the question file. For example, if the users defines a question file with selection answers “True” and “False” representing value ‘1’ and ‘0’, then the answer column in the MTurk answer table will have value either ‘1’ or ‘0’ stored in string data type. However, these may not be the value that users want the active learning module to learn from. In the above example, users may want to have Boolean data instead of string. In order to parse answers into labels, users can implement a function def
answer2Label (self, answer). Similar to the sample2info function, the graphical user interface provides a code editor for user to implement it. For example, if users want the answer ‘1’ to be True and ‘0’ to be False, they can implement answer2Label in the editor as shown in Figure 14.

![Python code editor for users to implement the answers2Labels function](image)

**Figure 14. Python code editor for users to implement the answers2Labels function**

Once this function is implemented, the learning automation layer can parse answers to labels and thus the labeling and learning operations do not require any manual effort from the users.

4.1.4.3 Removing HITs

There are two ways for users to remove HITs on MTurk. One is the resetAccount command and another once is the deleteHITs command. The resetAccount command will approve any HITs that are not yet approved and then remove all HITs in an MTurk account and. The deleteHITs performs similar actions but only removes HITs specified in a success file. The framework has both methods implemented and allows users to use these functions through the graphical user interface.
4.2 Learning automation

4.2.1 Database designs for learning automation

4.2.1.1 Sample table

The sample table basically stores samples, features and labels of a project. Each project has its own sample table named as `<projectName>_sample`. Active learning model loads labeled samples and features from this table for learning and unlabeled samples for labeling.

**Table 6. The sample table design**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sample_id</td>
<td>serial</td>
<td>Unique serial id for each sample</td>
</tr>
<tr>
<td>sample</td>
<td>text</td>
<td>Sample parsed in text form</td>
</tr>
<tr>
<td>feature</td>
<td>text</td>
<td>Feature of the sample in text form</td>
</tr>
<tr>
<td>label</td>
<td>text</td>
<td>Label of the sample in text form</td>
</tr>
<tr>
<td>label_source</td>
<td>string</td>
<td>The source of the label. (e.g. The ID of a HIT)</td>
</tr>
<tr>
<td>for_testing</td>
<td>boolean</td>
<td>Whether the sample is used for control group validation</td>
</tr>
</tbody>
</table>

4.2.1.2 Execution record table

The execution record table stores the record of important events or status of the framework such as start of learning process, termination of learning process, update of a classifier etc. Each project has its own execution record table named as `<projectName>_execution_record`. Information stored in this table includes the setting of the learning automation layer and progress of learning at the moment of recording. Details stored in this table are used to display milestones and render charts for users to monitor the framework and learning progress.
Table 7. The execution record table design

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>record_id</td>
<td>serial</td>
<td>Unique serial id for each record</td>
</tr>
<tr>
<td>time</td>
<td>timestamp</td>
<td>Time of the record made</td>
</tr>
<tr>
<td>record_type</td>
<td>text</td>
<td>Type of the record</td>
</tr>
<tr>
<td>sklearn_setting</td>
<td>text</td>
<td>A dict in text form storing the configuration of scikit-learn used at the moment of recording</td>
</tr>
<tr>
<td>control_group_size</td>
<td>integer</td>
<td>Total amount of control group samples at the moment of recording</td>
</tr>
<tr>
<td>control_group_accuracy</td>
<td>float</td>
<td>The last control group validation results at the moment of recording</td>
</tr>
<tr>
<td>k_fold_accuracy</td>
<td>float</td>
<td>The last k-fold cross validation results at the moment of recording</td>
</tr>
<tr>
<td>deviation</td>
<td>float</td>
<td>The deviation of the last k-fold cross validation result at the moment of recording</td>
</tr>
<tr>
<td>labeled_amount</td>
<td>integer</td>
<td>Total amount of labeled sample at the moment of recording</td>
</tr>
<tr>
<td>payment</td>
<td>float</td>
<td>Total payment made so far at the moment of recording</td>
</tr>
</tbody>
</table>

4.2.2 Initialization of learning

At the very beginning of learning operation, the learning automation layer queries the database to load all samples into memory. Labeled samples are stored in two lists named \texttt{l\_train\_pool} and \texttt{l\_test\_pool} while unlabeled samples are stored in another list named \texttt{u\_pool}. The \texttt{l\_train\_pool} list stores those samples used for training and the \texttt{l\_test\_pool} list stores those samples used for control group validation. If \texttt{l\_train\_pool} is not empty, this layer will further check the
diversity of labels. There must be more than one type of labels before active learning can start. If the above requirements are not fulfilled, the learning automation layer randomly picks samples twice the amount of samples per iteration and uploads them to MTurk for labeling in order to initialize the l_train_pool. After the creation of HITs, it will periodically check their status and retrieve answers. The actual active learning process will be kicked off once the l_train_pool fulfills the diversity requirement. If the diversity requirement is still not fulfilled after retrieving answers of all HITs, the learning automation layer will keep creating new HITs the same amount as samples per iteration with new randomly picked samples until the requirement is fulfilled and then start the active learning process. For instance, if a user specify HITs per iteration to be 5 and questions per HIT to be 10, then samples per iteration is 5 * 10 = 50 and in the initialization stage, this layer will create 10 HITs with 2 * 50 = 100 randomly picked samples. If none of the samples has different labels, it will create another 5 HITs with 50 random samples until the labeled sample pool has at least two samples with different labels.

4.2.3 Sampling strategy
After the initialization stage, the active learning module has the ability of selecting samples that can improve the classification accuracy most. In each iteration of picking new samples to be labeled, the sampling strategy involves three major procedures:

1. The learning model makes prediction on all unlabeled samples
2. The framework calculates the entropy of each sample prediction
3. The framework chooses the samples with highest entropy to be upload to MTurk for labeling

After that, the selected unlabeled sample will be passed to MTurk connector for HIT creation (as mentioned in 4.1.3 HIT creation) and wait for answers (as
mentioned in 4.1.2.1 Checking HIT status and 4.1.2.2 Retrieving answers). After retrieving labels, both the sample pools and the sample table will be updated.

4.2.3.1 Making prediction on unlabeled samples

The first step of sampling involves making prediction on all unlabeled samples. The framework first call the learning model and use function predict_proba(self.u_pool) to make probability prediction on all unlabeled samples. This function will return a two dimensional list in which each row represents a sample and each column indicates the probability of a sample belongs to a category. For example, if predict_proba function makes prediction on a samples and returns \([a, b]\), it indicates that the model predicts that \(a\%\) the sample belongs to category A and \(b\%\) to category B.

4.2.3.2 Calculating entropy

Entropy is used to measure the level of improvement that a sample can make to the model accuracy. There is no point for a model to get more samples in which the model is already very confident in its prediction. In other words, the most desirable samples are those that the active learning model has least confidence in its prediction on. Comparison on the levels of confidence can be easily achieved on two categories classification. For example, if the prediction on samples \(s_1\) and \(s_2\) function are \([\[a_1, b_1\], [a_2, b_2]\]) where \(a_1 > a_2\) and \(b_1 < b_2\), obviously the model is more confident on its prediction on the first sample. However, comparison on classification with three or more categories is not that simple. For example, it is hard to say which of the predictions is more uncertain if predict_proba function returns \([0.34, 0.32, 0.33], [0, 0.49, 0.51]\]). In order to make scientific comparison, Boltzmann’s H-theorem is adopted by the framework to calculate entropy:

\[
H(X) = - \sum_{i=1}^{n} P(x_i) \log_b P(x_i)
\]
In this formula, \( P(x_i) \) represents a single probability, \( n \) is the number of probabilities and \( b \) is the base of the logarithm. For this framework, \( P(x_i) \) is a single value in the list returned by `predict_proba`, \( n \) is the number of categories and \( b \) is the number of samples involved. The above formula can be transformed into the following format that is easier for programming:

\[
H(X) = -\sum_{i=1}^{n} P(x_i) \log_{10} P(x_i) / \log b
\]

Noticed that \( P(x_i) \) sometime can be zero and cause error, zero values are ruled out beforehand. If there is only one non-zero value, implying that the classifier predicts that a sample is 100% belong to a class, simply skip the calculation (because \( \log b = \log 1 = 0 \) will cause divided by zero error) and return 1 as is entropy.

4.2.3.3 Selecting samples

4.2.3.3.1 High entropy samples

After calculating the entropy of all unlabeled samples, the unlabeled sample pool is sorted according to their entropy. The sorting is in reverse order so the sample with the highest entropy is at the end of the list and can be pop out easily.

Theoretically, in the sorted unlabeled sample list, its tail contains the most desirable samples. However, during the development of the framework, it was discovered that these samples should not be selected successively. This is because there is a high chance that the samples with similar entropy have very similar features. For example, if the samples are RGB values, then entropy of \([1, 0, 0] \), \([0, 1, 0] \) and \([0, 0, 1] \) will be very similar. Selecting all of them cannot improve the model more than selecting only one of them. Moreover, it is unlikely that workers can distinguish the difference of these samples. Thus, selecting all of them will be a waste of time, money, and computation power.
The framework currently solves this problem by defining a range of samples considered as having high entropy and then selects samples in this range randomly. The range size in default is the minimum value of “forty times of samples per iteration” and “one tenth of the total amount of unlabeled samples”. For example, if users specified HIT per iteration to be \( m \), questions per HIT to be \( n \), then after sorting, the framework randomly select \( mn \) samples from the top \( 50mn \) samples. If the total amount of unlabeled samples is \( k < 50mn \) than, then the framework will select samples from the top \( k/10 \) samples instead of \( 50mn \).

Furthermore, if one tenth of the total amount of unlabeled samples is less than samples per iteration, the learning process will stop since it does not have sufficient unlabeled samples. Users must provide more samples if they want to start new iteration of learning.

### 4.2.3.3.2 Control Group Samples

Besides selecting samples according to their entropy, users can specify a ratio of samples to be selected randomly as the control group samples. They are mainly used for control group validation and the compensation of k-fold cross-validation (please see 4.2.4.1 K-Fold Cross-Validation and 4.2.4.2 Control Group Validation for more details about how these samples are used and the reason of having them). These samples are used for training as well. Although they cannot improve the classification accuracy as much as those high entropy samples, they can make the classifier more all-round. Configuration of Control Group Radio is an advanced option under the active learning and MTurk setting. In default, its value is 50, meaning that the amount of control group samples is 50% of the amount of high entropy samples. For example, if users specify samples per iteration to \( S \) and Control Group Radio to be \( c\% \), then each HIT consists of \( S/(1 + c/100) \) high entropy samples and \( S - S/(1 + c/100) \) control group samples. They will be shuffled before being uploaded to MTurk so that workers will not be able to distinguish between control group samples and high entropy samples.
Control group samples are stored separately from high entropy samples. In the pending label stage, high entropy samples are stored in `p_train_pool` while control group samples are stored in `p_test_pool`. These two pending pools are in `dict` data type with sample ID as the key so when a label is retrieved, the framework can easily and efficiently map labels to samples. After mapping, high entropy samples and control group samples are moved correspondingly to `l_train_pool` and `l_test_pool`, which are `list` data type. In the database, there is an integer value named `for_testing` used to indicate whether a sample is in the control group (1 for yes and 0 for no) so when samples are loaded from database to memory, control group samples can be identified and handled separately.

4.2.4 Accuracy evaluation

4.2.4.1 K-Fold Cross-Validation

K-Fold Cross-Validation is the most common approach to evaluating the accuracy of a classification model. In this framework, 5-fold cross-validation is implemented for user to assess the learning result. In 5-fold cross validation, labeled samples are randomly split into 5 equal size groups. Each time 4 groups are used for training a temporary classifier used to make prediction on the remaining group of samples, then the framework compares the predicted classes with the actual labels and counts the percentage of correct prediction. The above operation repeats 5 times so all 5 groups of samples have been used for testing. However, a classifier can only be built when there are at least two samples labeled as different categories (as mentioned in 4.2.2 Initialization of learning). Thus, k-fold cross validation is applicable only when there are enough labeled samples in different categories. For 5-fold cross validation, the following requirements must be met:

1. There must be at least 5 labeled samples
2. There must be at least 2 samples having different class(es) to others
3. Any selection of 4 groups out of 5 must meet the diversity requirement.

For example, if the labeled sample pool is \([A, A, A, B, B]\), then we can group samples into \([A], [A], [A], [B], [B]\) so any 4 of them can meet the diversity requirement, which is not achievable by \([A, A, A, A, B]\) (failing in requirement 2). If the labeled sample pool is \([A, A, A, A, A, A, A, A, B, B]\), the only available grouping is \([A, A], [A, A], [A, A], [A, B], [A, B]\) because it is the only way to fulfill requirement 3.

In the actual implementation, the framework checks the requirement 1 and 2 at the start of the validation stage. If they are fulfilled, random grouping will start to split samples into five equal size groups. Otherwise it will skip the validation stage. After grouping, the framework further checks for requirement 3. If requirement 3 is not fulfilled, it skips the validation stage. The framework does not enforce requirement 3 even if there exists an eligible grouping solution. For example, the framework may group \([A, A, A, A, A, A, A, A, A, B, B]\) into \([A, A], [A, A], [A, A], [A, A], [B, B]\) but not \([A, A], [A, A], [A, A], [A, B], [A, B]\) which is the only eligible grouping. This is because failing to fulfill requirement 3 usually happens at the early stage of active learning when there are only a few labeled samples available in the pool. The accuracy evaluation will not be significant at this moment anyway. More importantly, grouping should be done randomly so that the evaluation result can reflect the expected value of accuracy of the model. It is not appropriate for the framework to intervene it. If requirement 3 is fulfilled as well, 5-fold cross validation will start by iterating the train-predict-compare operations 5 times. The average score will be the expected accuracy provided to users while the standard deviation is given by doubling of the standard score. After the evaluation process, the accuracy and deviation value are recorded to the record table in the database.

Although k-fold cross validation is the most common way of evaluating the accuracy of a classification model, it does not have much reference value if it involves only high entropy samples. In of active learning, samples are picked to
be labeled according to their entropy. In other words, existing labeled samples were picked because the classifier was unsure of its prediction at that moment. So when labeled samples are split into 5 groups, a classifier trained with 4 groups will not be able to make confident and accurate predictions on the remaining group of samples. As a result, k-fold cross validation involving only high entropy samples always greatly underestimates the accuracy of the model.

To solve this problem, control group samples are selected (see 4.2.3.3.2 Control Group Samples) to perform control group validation as an alternative to provide a more accurate evaluation. Control group samples are also added to k-fold cross validation as the compensation for the uncertainty caused by high entropy samples. Since these samples are picked randomly, classifiers usually can give better confidence and higher accuracy on its prediction. Involving these samples in k-fold cross validation can make the evaluation result closer to its actual accuracy. Therefore, instead of performing 5-fold cross validation on the $l_{train\_pool}$, the framework first temporarily merges the two labeled sample pools ($l_{train\_pool}$ and $l_{test\_pool}$) together, then randomly splits it into 5 equal sized groups and performs the rest of the operations.

4.2.4.2 Control Group Validation

Since the k-fold cross validation greatly underestimates the accuracy of classification models, the framework has control group validation implemented in order to provide user a better assessment on the learning results. As mentioned in 4.2.3.3.2 Control Group Samples, control group samples are stored in $l_{test\_pool}$ separated from high entropy samples which are stored in $l_{train\_pool}$. In control group validation, a temporary classifier is trained with only those high entropy samples. Then this classifier is used to make prediction on the control group samples in $l_{test\_pool}$. This validation methods still underestimate the actual accuracy of the classifier since the actual classifier that the framework exports to users is trained with all labeled samples including both $l_{train\_pool}$ and
However, unlike the case of 5-fold cross validation that missing one fifth of the high entropy samples can cause serious drop of accuracy, missing \( l_{\text{test\_pool}} \) in the training of temporary classifier does not affect its accuracy much. As a result, the accuracy given by control group validation can be very close to the actual accuracy.

4.3 Project and client application creation and execution

4.3.1 Database design for project and client application

4.3.1.1 Project table

The project table stores information of each project, including MTurk setting, scikit-learn setting and user-defined functions. The generation of client application replies on the details stored in this table. Project name stored in this table is also used for creating other tables, such as \(<\text{projectName}>_{\text{pending}}\) table. The detailed table structure is as follows:

**Table 8. The project table design**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>text</td>
<td>Unique name of the project</td>
</tr>
<tr>
<td>description</td>
<td>text</td>
<td>Description of the project</td>
</tr>
<tr>
<td>useMTurk</td>
<td>integer</td>
<td>An integer indicating the status of MTurk setting</td>
</tr>
<tr>
<td>MTurkDetail</td>
<td>text</td>
<td>A <code>dict</code> in text form storing the MTurk configuration</td>
</tr>
<tr>
<td>sklearnMode</td>
<td>text</td>
<td>Scikit-learning execution mode. Value can be SVM, Decision Trees, Naive Bayes, etc.</td>
</tr>
<tr>
<td>sklearnDetail</td>
<td>text</td>
<td>A <code>dict</code> in text form storing the configuration of scikit-learn</td>
</tr>
<tr>
<td>functions</td>
<td>text</td>
<td>The <code>samples2Info</code> function and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>answers2Label function in text form</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>has_change</td>
<td>integer</td>
<td>An integer indicate whether the setting have been changed</td>
</tr>
<tr>
<td>sample_predict</td>
<td>string</td>
<td>The predicted sample distribution list in string format</td>
</tr>
</tbody>
</table>

### 4.3.2 Project Creation

The framework defines “project” as the creation and execution environment of a client application. In order to create a new project, users need to provide only a project name and description. Creation of a project involves three actions: (1) inserting a row in the *project table*; (2) creating `<projectName>_sample` table; (3) creating a folder named by the project name. The new row in the project table contains project name, description and *useMTurk* value. The *useMTurk* value is set to be 0, indicating that MTurk is not yet configured. Also, since the project name is defined as the primary key of the *project table*, the framework examines the uniqueness of the name provided by users and rejects any creation with existing project name. The only table needed to be created at the moment of project creation is `<projectName>_sample` table, since every project must need it. MTurk related tables such as `<projectName>_pending` and `<projectName>_answer` are created at the first execution of client application but not at the moment of project creation since the user may choose not to use MTurk and thus these tables may not be needed. Finally, a folder named by the project name is created under the project directory. It is used to store *properties file*, *question files*, client application’s `.py` file and other files.

A newly create project cannot generate and execute client application immediately. Some configuration must be made beforehand.
4.3.3 Client application creation pre-requisites

4.3.3.1 Active learning and MTurk setting

Before a client application can be generated, users must configure MTurk setting via the graphical user interface as shown in Figure 16.

They can choose not to use MTurk labeling and that updates the value `useMTurk` stored in `project table` to 2. Then the learning module will only learn from labeled samples provided by the users. If users choose to use MTurk for labeling, this
value will be set to 1. If that’s so, they must further provide values for AWS Access Key, AWS Secret Key, HITs per iteration, threshold, and interval. As mentioned in 4.1.2 MTurk connection, AWS Access Key and AWS Secret Key are necessary for accessing MTurk server. The rest parameters are used by the learning automation layer. HITs per iteration defines how many HITs should be created in each iteration of model upgrading; threshold defines a value in case if the amount of available pending HITs on MTurk server is lower than this value, than the framework should start a new iteration and create new HITs; interval is a time span in second indicating the time interval between each MTurk checking.

After basic setting, users must further configure properties file and question files, as mentioned in 4.1.3.1 Properties file and 4.1.3.2 Question file. After form submission, properties file and question file will be created and stored while the basic setting will be parsed into dict data type and stored in the project table in text form, like the following:

```
{
    'intervalInSeconds': '60',
    'HITsPerIteration': '5',
    'AWSAccessKeyId':
        'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx',
    'threshold': '2',
    'AWSKey':
        'xxxxxxxxxxxxxxxxxxxxx'
}
```

When this information is needed, the framework uses literal_eval(<data in text form>) to transform text back to dict data type.

4.3.3.2 Scikit-learn setting

Similar to MTurk setting, users must configure scikit-learn beforehand with the form provided in the graphical user interface. Values required are the parameter that learners of scikit-learn need. Default values are provided for all fields so that users can simply click save to finish configuration if they do not have specific requirements. Advanced options is hidden and only displayed on request to avoid misuse since they require certain knowledge in the learning algorithm.
Since the active learning mechanism is based on the comparison of prediction entropy, the classifier must have probability prediction support. The framework currently supports 5 classification models with total 13 type of classifiers:

Table 9. Supported classification models and classifier types

<table>
<thead>
<tr>
<th>Models</th>
<th>Classifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support Vector Machines</td>
<td>C-Support Vector Classification</td>
</tr>
<tr>
<td></td>
<td>Nu-Support Vector Classification</td>
</tr>
<tr>
<td>Decision Trees</td>
<td>Decision Tree Classifier</td>
</tr>
<tr>
<td></td>
<td>Extremely Randomized Tree classifier</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>Gaussian Naive Bayes Classifier</td>
</tr>
<tr>
<td></td>
<td>Naive Bayes Classifier for multinomial models</td>
</tr>
<tr>
<td></td>
<td>Naive Bayes Classifier for multivariate Bernoulli models.</td>
</tr>
<tr>
<td>Nearest Neighbors</td>
<td>K-Nearest Neighbors Classifier</td>
</tr>
<tr>
<td>Ensemble methods</td>
<td>AdaBoost Classifier</td>
</tr>
<tr>
<td></td>
<td>Bagging Classifier</td>
</tr>
<tr>
<td></td>
<td>Extra-trees Classifier</td>
</tr>
<tr>
<td></td>
<td>Gradient Boosting Classifier</td>
</tr>
<tr>
<td></td>
<td>Random Forest Classifier</td>
</tr>
</tbody>
</table>

The framework provides a page for users to select a classifier and configure it in depth. It enables the configuration of all available parameters for the above classifiers except some python object type parameters for Ensemble methods.
Figure 17. Page for configuring scikit-learn

Same as MTurk setting, these values will be parsed into dict data type and stored in the project table in text form:

```python
{'kernel': u'linear', 'C': u'1.0', 'verbose': u'False', 'probability': True, 'max_iter': u'linear', 'shrinking': u'True', 'class_weight': None, 'degree': 3, 'gamma': 0.0, 'coef0': 0.0}
```

4.3.3.3 User function setting

Users are required to implement two functions before a client application can be created. Details of the function `sample2Info` can be found in 4.1.3.3 Input file and details of the function `answer2Label` can be found in 4.1.2.2 Retrieving answers. Similar to MTurk setting and scikit-learn setting, the implementation of these two functions are stored in project table in text form. The framework first treats the two functions as strings and stores them in a dict object, then parses the dict object into text. After the above operation, the following text will be stored in the project table:
4.3.4 Client application creation

Before the generation of client application, the framework will check the existence of MTurk setting, scikit-learn setting and user-defined functions in project table, and the existence of properties file and question file in the project directory. If any of the above is missing, the framework will disable any generation request as shown in Figure 18.

```python
{'samples2Info': u'def sample2Info(self, sample, feature):
    info = []
    global count
    if count is None:
        count = 0
    from PIL import Image
    for sample in samples:
        im = Image.new('RGB', (300,300), tuple(sample))
        im.save('/home/xxxxxx/Dropbox/Public/color'+str(self.count)+'.jpg')
        data.append('https://dl.dropboxusercontent.com/u/xxxxxxx/color'+str(self.count)+'.jpg')
        count = count + 1
    return info

'answer2Label': u'def answer2Label(self, answer):
    labels = []
    #Please put your code here
    for answer in answers:
        if answer[0] == '0':
            labels.append(False)
        elif answer[0] == '1':
            labels.append(True)
    return labels
```
process. Each classifier type has their own template defining how they call scikit-learn modules and the learning automation layer. Each template has eight placeholder: \{0\} for project name, \{1\} for active learning and MTurk setting in \texttt{dict} data type, \{2\} for scikit-learn setting in \texttt{dict} data type, and \{3\} for user-defined functions, \{5\} for MTurk connector object, \{6\} for automated learning function, \{7\} for clearing MTurk function. The first four elements already contain all the parameters required to perform automated active learning while the rest are subjected to whether users enable MTurk or not. All other functions call the \texttt{dict} object to obtain required value. For example, in the initialization of MTurk connector object, \texttt{aws_key} parameter is assigned by looking at \texttt{mturk dict} using \texttt{aws_key = mturk['AWSKey']}. Template rendering for project name, mturk setting and scikit-learn setting is fairly simple. In scikit-learn setting and MTurk setting (not including properties file and question file), only \texttt{AWS Key} and \texttt{AWS Secret Key} are strings inputted by users (all other inputs are restricted to numeric or selection types) while proper \texttt{AWS Key} and \texttt{AWS Secret Key} never contain characters like \{"\} and \{'\}. Therefore, it is safe to use \texttt{.format()} function without checking the need of escape. Also, since mturk setting and scikit-learn setting stored in \texttt{project table} are already in \texttt{dict} data type parsed to text, the whole string can be directly used to replace the placeholders using \texttt{.format()} function.

For function \texttt{samples2Info} and \texttt{answers2Label}, they cannot be used to replace placeholder directly. The line separator used by HTML input fields is ‘\r\n’, which is inconsistent to Python’s line separator, which is ‘\n’. Also, what users provide are the implementation of abstract functions defined in superclass \texttt{learning_automation} (which is the learning automation layer). Since Python use indentation instead of curly braces to delimit blocks, each line of the \texttt{sample2Info} and \texttt{answer2Label} must have an increase in indentation. In view of this, the framework first splits the two functions by ‘\r\n’, then add proper amount of spaces in front of each line, and finally joins each line with ‘\n’.
After the template rendering operation, an `execute.py` file is generated and stored under the project directory.

### 4.3.5 Client application execution

After the generation of `execute.py`, users can start the execution of client application. Execution of client applications involves uploading samples to the database and defining execution environment.

![Figure 19. Page for user the upload samples and execute client applications](image)

#### 4.3.5.1 Uploading samples

Samples going to be uploaded to database must be a CSV file containing four columns with tab `\t` as the column separator. The first column is the samples; the second column is features; the third column is label (use `NULL` if the sample has no label); and the last column is whether the sample is for control group validation (1 for yes and 0 for no). The CSV does not need to have sample ID column since sample ID in the database is set to be serial type, which will be generated automatically. Meanwhile, features must be in list data type parsed to text form. For example, if samples are `RGB value` and features are `(\<RGB value\>/256)`, then the CSV file should contain:
On the client side, the form uses `enctype="multipart/form-data"` to enable submission of file type. On the server side, the file is extract with `request.files['samplesCSV']`. Insertion of data from CSV file to sample table can be achieved by using function `copy_from(<file>, <tableName>, <columnName>)` provided in psycopg2 library. The framework will extract the amount of rows (samples) inserted and display to the user. If exception is caught during the operation, `rollback()` will be called to ensure that the sample pool is not damaged. Users can upload samples at any time even during the execution of auto-learning. However, samples will only be loaded to memory once at the start of client application execution. Therefore, if users want the newly uploaded samples be involved in the learning process, they must restart their application (current learning progress will not get lost).

### 4.3.5.2 Execution

In order to start execution, users must specify whether to use MTurk sandbox (as mentioned in 4.1.2 MTurk Connection), and the criteria of ending the learning process. The users can choose to stop the learning process when the accuracy calculated by K-folder validation or control group validation is higher than a certain value, or when the amount of labeled samples involved in learning achieve a certain value. The former criteria is optional but the later criteria is compulsory in order to prevent users from setting an unreachably high accuracy causing infinite labeling and learning.

Execution of client application is achieved by importing the previously generated `execute.py` at runtime with `imp.load_source()` function and calls its `execute(useMTurk, errorRate, upperBound)` function. It is done by the web-application (server-side) layer. It will open a new thread for executing the client
application. The web application (server-side) layer maintains a global variable named `caThreads` in `dict` data type. Each `Thread` object for client application execution is added to `caThreads` with a key same as the project name. `caThreads` is also used to check whether the client application is already in execution by checking the existence of `projectName` in `caThreads` and whether that thread is alive. When the execution is completed, `Thread` object will be removed from `caThreads`.

4.4 Web application interface

4.4.1 Structure of the web application (server-side) layer
Web application layer has three main directories:

- **View**: This directory stores `.py` files that control the web application, such as handling client requests, rendering templates, making responses, accessing database, calling labeling automation layer and client applications.

- **Template**: This directory stores `.html` files that contain templates of pages which are going to be rendered and displayed to clients.

- **Static**: This directory stores `.css`, `.js` files and some images used by `.html` template.

The structure of the web application (server-side) layer can be illustrated as Figure 20.
Figure 20. Structure of the web application (server-side) structure
4.4.1.1 General request control

Each .py file stored in the view directory is considered as a module of the web server. Each of these modules are assigned to handle a specific region of requests while a region is defined by requests’ URL prefix. Within each module, each request to a specific page is handled by a specific function. By registering all modules to the application level object, the server is formed to handle requests to all URLs with proper page rendering and redirection:

In order to prevent break-in access (users directly enter a URL to skip the framework setup procedures), a function _render_template() is implemented and used for every rendering operation instead of the render_template() function provided by Flask. The _render_template() function calls needSetup() function to check whether the framework is set up already. Only the setup page, about page and welcome page can be access without setting up. If the framework is setup already or the requested page is one of the above, _render_template() will call render_template() to render the requested page. Otherwise, users will be redirected to the welcome page.

In the render_template function, a parameter needSetup is passed to the page rendering process. It is used to hide the all other tab expect home and about tab in the top navigation bar. In the template contain navigation bar, needSetup parameter is picked up by Jinja2 syntax to make decision on which parts of a template should be rendered for users.

4.4.1.2 General template layout

There is a template named layout.html that contains the HTML and Jinja2 syntax of the navigation bar, as well as the CSS and JS that all templates need. In other words, the layout.html defines and implements the structure that every page must follow. All other templates are inherit from this layout.html by adding {% extends "layout.html" %} at its front. In the layout.html, there are three blocks defined in a way that templates inheriting from it can modify. One of the
modifiable block is the head block. In layout.html, the head block already includes all necessary header information and CSS. Another modifiable block is the endScript block. It is placed at the end of the layout.html for placing JavaScript related contents. Similar to the head block, necessary scripts are already implemented in layout.html. When other templates need to add extra details to the header or endScript block, they do the following:

{% extends "project/dashboardLayout.html" %}
{% block head %}
    {{ super() }}
    <!-- CSS for this specific template -->
    <style type="text/css" media="screen">
        ...
    </style>
{% endblock %}

The most important modifiable block is an empty block named content implemented as {% block content %}{% endblock %}. All template dependent contents are implemented in this block so they are displayed under the navigation bar. There is a secondary level layout template called dashboardLayout.html. This template inherit from layout.html but with further implementation that all project dashboard pages must follow, including CSS added into head block, JQuery added into endScript block, and a left side-bar added into content block. The dashboardLayout.html defines another block named dashboardContent inside the content block. All dashboard templates inherit from this dashboardLayout.html so they impliedly inherit from layout.html as well. However, their contents must be inserted to the dashboardContent block defined in dashboardLayout.html instead of the content block defined in layout.html. Figure 21 provides an illustration on the relationship between the layout.html, content block, dashboardLayout.html and dashboardContent block.
4.4.2 Active learning performance and MTurk monitoring

4.4.2.1 Line chart for Active learning performance monitoring

In order to allow users to monitor the learning performance such as model accuracy, samples involved and payment, a line chart (as shown in Figure 22) is implemented to visualize the performance of active learning.
The line chart is a scalable vector graphics (SVG) rendered using JQuery with d3.js support. It takes iteration number as the x-axis and accuracy in percentage as y axis. There are four lines on the chart. The one in black corresponds to control group validation accuracy. The other three lines in blue color from top to bottom are the upper 5-fold cross validation accuracy, average 5-fold cross validation accuracy, and lower 5-fold cross validation accuracy. Area between upper and average accuracy represents the optimistic expectation while area between lower and average accuracy represents the pessimistic expectation.

4.4.2.1.1 Data retrieval
When the statistic page is opened, client side JQuery scripts first try to query the server for accuracy data via a URL: `<projectName>/statistic/_getAccuracyData`. On the server side, a function `def _getAccuracyData(projectName)` is implemented to handle requests to this URL. This function query the database’s execution record table to obtain all necessary data and parse them to proper JSON format then return it like the following:

```json
{"data": [
    {"iteration": 0, "deviation": 0, "kFoldAccuracy": 0, ...},
    {"iteration": 1, "deviation": 0.417, "kFoldAccuracy": 0.312, ...},
    ...
]}
```

On the client side, JQuery scripts pick up the response and further parse to proper format that the line chart uses. If the client side successfully obtain accuracy information, it further queries the server for marker information which indicates the start, end or termination of client application execution. It is done by requesting URL: `<projectName>/statistic/_getExecutionMarkers`, which is handled by `def _getExecutionMarkers(projectName)`. After getting all necessary data, the rendering of the line chart will start.
4.4.2.1.2 Chart construction

The size of the chart is set to be relative to the window size while all elements within the chart are set to be relative to the chart size. The whole chart is constructed with multi-level dependence. First of all, an empty SVG object with unique id is appended to an empty div block. Then it calculates the length and density for x-axis and y-axis. After the calculation, they are appended to the root SVG object. Meanwhile, line objects are added according to the position and density of x-axis and y-axis to construct the grid in the chart background. For accuracy lines, they are rendered as cardinally interpolated line objects. Similarly, accuracy areas are rendered as cardinally interpolated area objects.

For those markers on the chart, they consist of a line object and a circle object. There are four types of markers available: start of execution, finish of execution, execution terminated by users, and execution terminated because of errors. They are assigned to different classes and distinguished by their labels and colors controlled by CSS. The length of line objects is the same as the height of the chart while the y position of circle objects is the chart height plus its radius so they are placed on the top of the chart. Their x position is obtained from the x-axis scale attribute according to the iteration number it belongs to.

4.4.2.1.3 On-hover tooltips

On-hover tooltips is added to the chart for users to obtain information such as time of the iteration, control group size, the actual accuracy value, total payment so far and etc. The on-hover tooltips comes with a vertical line indicating with iteration users are hovering on as shown in Figure 23.
All tooltip elements (including all text objects, line objects and rectangle objects etc.) are added to an array named `tooltipElement` so they can be handled in once. An invisible rectangle object with size same as the chart area is added for catching on-hover events. The on-hover event is actually controlled by three events: `mouseover`, `mouseout` and `mousemove`. When a `mouseout` event is caught, all tooltip elements will be set to display none.

When a `mouseover` or `mousemove` event is caught, all tooltip elements will be rendered to a proper position and display information according to the iteration that is closest to cursor position on the chart. In order to distinguish the closet iteration to user’s cursor, firstly the cursor position in pixel relative to the browser window are parsed to the x and y position in the chart. Secondly, it calculates the x position of each iteration in the chart and the y position of the accuracy line for that iteration. By comparing the x position of the cursor to the x position of each iteration, the closest iteration can be found. Then the tooltip elements’ origin position will be transformed to the x position of that iteration and the highest accuracy line’s y position. For text elements, the content will be updated to the information of that iteration.

4.4.2.1.4 Animations and effects
The accuracy line chart contains three animations and effects:

1. The growth of accuracy lines and areas
2. The appearance of markers

3. The raise of markers

All accuracy lines and areas are controlled by a path object in order to control CSS style and the first effect. Also, they are linked to a clipPath object used to control the animation effect that the accuracy lines grows from left to right hand side. The clipPath object contains a rectangle object at the same layer as lines and areas. Only objects overlapped by it are visible. When all elements of the chart are in place, its width is enlarged from width = 0 to width = chartWidth within 2500 milliseconds using the transition() and duration() functions to reveal the accuracy lines and areas. This operation is like drawing a curtain open while the resulted effect is like the growth of accuracy lines and areas.

For markers’ appearance, in order to coordinate the animation effect of accuracy lines and area, each marker is added to the root SVG object according to their time sequence with a time interval so that they appear one by one from left to right. Moreover, markers stretch out from the bottom to the top by controlling the y1 position of their line objects and the y position of their circle objects with a path object, transition() function and duration() function within 1000 milliseconds. The following is an illustration of the timeline of value changes:

![Figure 24. Line chart animation timeline](image)
4.4.2.2 Pie charts for sample distribution and MTurk performance monitoring

In order to allow users to monitor the samples and MTurk performance, a customized SVG pie chart is implemented in this project for the framework to display radio information as shown in Figure 25.

**Figure 25. Pie chart of duration rate**

For samples and labels monitoring purposes, pie charts are used for visualize label distribution, predicted sample distribution and overall sample distribution. For MTurk performance monitoring, information includes answered rate (radio of answered HITs and expired HITs), pending time (distribution of time from HITs being created to HITs being accepted), and duration rate (distribution of time that workers spent on completing HITs). Users can use these information to review the effectiveness and attractiveness of their HIT design so they can accordingly adjust HIT contents and rewards.

4.4.2.2.1 Data retrieval

Data gathering of pie chart uses the same strategy as the line chart that JQuery queries server for JSON data via specific URLs. The JSON data contain two attributes: description and rate. Description is a string to be displayed as a label...
and it can be in HTML format. Rate is a float number indicating the percentage of that partition. For example, it could be as follows:

```
{"data": [
    {"description": "<b>7</b> HITs are ...", "rate": 0.175},
    {"description": "<b>17</b> HITs are ...", "rate": 0.425},
    {"description": "<b>5</b> HITs are ...", "rate": 0.125},
    ...
]}
```

Data for different pie charts come from different tables in the database. For label distribution, data are instantly gathered from 4.2.1.1 Sample table. For predicted sample distribution, data are generated after each iteration of learning and stored in 4.3.1.1 Project table under the sample_predict column. Data of overall sample distribution are actually a combination of the above two data sources. For MTurk performance monitoring, all data come from 4.1.1.2 MTurk Record table. Answered rates are counted using the information in the is_expired column; pending time rates are calculated using the information in the creation_time and accept_time columns; and duration rates are calculated using the accept_time and submit_time columns.

However, unlike sample distribution which has an explicit amount of classes as the partitions for pie charts, MTurk performance information is scattered numbers (e.g. different HITs have different pending time). To obtain a suitable amount of partitions, a function grouping(data) is designed to transform scattered numbers to number ranges. It first sorts all numbers and calculates their differences to their previous one. Then it computes the average value of differences. Primary grouping is done by adding numbers to a group successively until a number has a difference larger than the average difference, then it opens a new group. After this operation, there is a chance that some groups have only one number, so secondary grouping is needed. If these groups are successive, they are merged together into one group. If they are isolated, they will be appended to the closest adjacent group. Finally, if there are still too many groups, the
framework will merge the closest groups together. It calculates the average value of each group and computes the differences of the average values of adjacent groups. Adjacent group pair with minimum average values difference will be merged until the limitation is met (in default the boundary is 5 groups).

**4.4.2.2.2 Pie chart rendering**

A pie chart actually consists of pie, arc, circle and line objects. If all objects are no filled, the pie chart in Figure 25 will look like the following:

![Pie chart diagram](image)

**Figure 26. The skeleton of pie chart in Figure 25**

In a pie chart, each partition is represented by a pie object. A filled center circle (shown as a black circle in Figure 26) covers these pie objects partially so that they look like arches on a chart. The first pie object has start angle equal to 0 degree and its end angle is calculated using its rated value. For the rest of pie objects, their start angles are the end angles of their previous pie and their end angles are their start angles plus their rates in degrees.

Each label and description has a poly line object that links them to the pie that they belongs. Each poly line object has three points: \( p_0(x_0, y_0), p_1(x_1, y_1) \) and \( p_2(x_2, y_2) \) while \( p_2 \) has y coordinate \( y_2 = y_1 \) and x coordinate \( x_2 = x_1 + r \) where
$r_p$ is the radius of pies (if the pie is on right hand side) or $x_2 = x_1 - r_p$ (if it is on left hand side). In order to display labels, descriptions and poly lines with position relative to their parent pies, two invisible circles are added (shown as grey dotted circles in Figure 26). The inner one has a radius $r_i = (r_p + r_c)/2$ where $r_c$ is the radius of the center circle. The outer one has a radius $r_o = 1.25r_p$. Poly line objects always have $p_0$ on the border of the inner invisible circles and $p_1$ on the border of the outer invisible circles. By transforming the pie center to the origin and treating $d = (\text{start angle} + \text{end angle})/2$ as the angle, coordinates of $p_0$ and $p_1$ can be obtained using function $\text{arc.centroid}(d)$. To determine whether a pie is on the left or right hand side, it compares the middle radian angle to $\pi$ by considering $(\text{start angle} + \text{end angle})/2 < Math.PI ? \text{onRight: onLeft}$. The position of label and HTML description depends on $p_2$. For y position, label is on $y_2 + \text{test height}$ while HTML description is on $y_2 + 100$. For labels’ x position, it is equal to $x_2$ if it is on the left hand side or $x_2 + \text{text width}$ if it is on the right hand side. Descriptions’ x positions depend on $x_1$ instead of $x_2$. HTML descriptions are actually foreign objects rendered as div blocks with width set to the length of $p_1p_2$, which is $|x_1 - x_2|$, and their x position is always set to be $x_1$. If their parent pies are on left hand side, they are set to be $\text{text-align: left}$. Otherwise, they have $\text{text-align set to right}$.

The center circle contains the chart title and chart description. In order to automatically adjust their size to fit the center circle, text objects instead of foreign objects are used. However, the SVG text object does not have line breaks function. A customized auto line breaking method $\text{autoNewLine}(\text{caller}, \text{width}, \text{sizeRadio})$ is designed. This function automatically extracts text objects within the caller object, calculates the width of characters according to the sizeRadio, splits each text object to multiple $\text{tspan}$ objects such that their width is less than or equal to the width parameter, and adjusts their y positions properly.
4.4.2.2.3 Animations and effects

Pie charts contain four animations and effects:

1. The appearance of pies
2. The appearance of the center circle
3. The appearance of poly lines
4. The transition of percentage labels

The appearance of pies is an animation of growing pies like opening a folding fan. It is achieved by controlling the start angles and end angles using `transition()` and `duration()`. All start angles and end angles have initial value 0 and then are increased to their actual value within 1500 milliseconds. The transition values are calculated using `interpolate()` function. At the meantime of pie growing, the center circle grows as well to make pies become arches. This effect is achieved by increasing the radius of the center circle from 0 to its real radius. The duration of center circle growth is the same as pies, which is 1500 milliseconds. After the completion of growth, chart title and description are displayed within it.

After the appearance of pies and the center circle, poly lines are displayed with sequence \( p_2, p_1 \) then \( p_0 \) so they appear far away from pies and then extends and plugs to pies. This effect is achieved with the use of a two-stage transition on coordinates. Initially, \( p_0 \) and \( p_1 \) are set to the coordinate of \( p_2 \). In the first 750 milliseconds, x coordinate \( p_0 \) and \( p_1 \) moves from \( x_2 \) to \( x_1 \) while the y coordinate remains unchanged. So at \( t = 750 \), \( p_1 \) is on its real coordinate overlapping \( p_0 \). For \( t = 750 \) to \( t = 1500 \), \( p_0 \) moves from \( (x_1, y_1) \) to its real coordinate \( (x_0, y_0) \).

Percentage labels have number increasing effect. Percentage labels start from 0.0 % and increase to their targeted value in 1500 milliseconds. A customized interpolator is implemented. It first gets the targeted value \( v \) and calculates the step for each millisecond and get \( v_t / 1500 \). Then it parse the current text to get the current value \( v_c \), and replace the text by \( v_c + v_t / 1500 \).
The following is an illustration of the timeline of value changes:

![Timeline Illustration](image)

**Figure 27. Pie chart animation timeline**

### 5. Results

#### 5.1 Labeled samples amount and model accuracy

##### 5.1.1 Experiment design

Experiments were designed to examine the framework performance on active learning and MTurk labeling. The goal was to explore how many labeled samples the framework could reduce for model training and how great the framework is able to assess model accuracy. Experiments were carried out using two pre-labeled dataset which each of them had around 5500 samples.

In treatment experiments, for each dataset, the framework performed active learning with MTurk labeling. When the amount of labeled samples met 200, 300, 400, 500 and 600, the control group validation accuracy and 5-fold validation accuracy were noted down. Also, a classifier was exported to make prediction on all samples of the dataset. Prediction results were compared to the actual label to obtain the actual accuracy of the classifier.

For control experiments, 200, 300, 400, 500 and 600 random labeled samples were picked and imported to the framework. The framework performed passive learning (by disabling active learning and MTurk features) with the same
machine learning algorithm and configuration. The resulted classifiers were exported and used to make predictions on all samples of the dataset. Prediction results were compared to the actual label to obtain the actual accuracy of classifiers. If the classifier trained by passive learning with 600 random samples gives lower accuracy than the classifier trained by active learning and MTurk labeling with 600 samples, the amount of random samples is increased until it meets similar accuracy.

Moreover, since the accuracy of a classifier greatly depends on sample selection, in order to obtain convincing results, both treatment experiments and control experiments were carried out three times for each samples amount. Comparison were made on the average accuracy to improve credibility.

5.1.2 Experiment on the optical recognition of handwritten digits data set
The optical recognition of handwritten digits data set was obtained from UCL machine learning repository. Samples are handwritten digits gathered from 43 people. Digits are in a form of 32x32 bitmaps and transformed to png files by setting the sample2Info function. Its composition is as follows:

<table>
<thead>
<tr>
<th>Class</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>554</td>
</tr>
<tr>
<td>1</td>
<td>571</td>
</tr>
<tr>
<td>2</td>
<td>557</td>
</tr>
<tr>
<td>3</td>
<td>572</td>
</tr>
<tr>
<td>4</td>
<td>568</td>
</tr>
<tr>
<td>5</td>
<td>558</td>
</tr>
<tr>
<td>6</td>
<td>558</td>
</tr>
<tr>
<td>7</td>
<td>566</td>
</tr>
<tr>
<td>8</td>
<td>554</td>
</tr>
<tr>
<td>9</td>
<td>562</td>
</tr>
<tr>
<td>Total samples: 5,620</td>
<td></td>
</tr>
</tbody>
</table>

Details on https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

Machine learning algorithm chosen for this dataset was Support Vector Machine, which is commonly adopted for image recognition. Its detail configuration is shown in Table 11. For active learning and MTurk labeling, sampling range was set to be 40 folds of samples per iteration and control group radio was set to be 50%. Detail configuration is shown in Table 12.
Table 11. Experiments on digits OCR data set: machine learning configuration

<table>
<thead>
<tr>
<th>Major machine learning configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
</tr>
<tr>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>Classifier type</td>
</tr>
<tr>
<td>SVC</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>1.0</td>
</tr>
<tr>
<td>Kernel</td>
</tr>
<tr>
<td>rbf</td>
</tr>
<tr>
<td>gamma</td>
</tr>
<tr>
<td>(1/n\text{_features} = 0.1)</td>
</tr>
</tbody>
</table>

Table 12. Experiments on digits OCR data set: Active learning and MTurk configuration

<table>
<thead>
<tr>
<th>Active learning and MTurk configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enlarge sampling range</td>
</tr>
<tr>
<td>40 folds</td>
</tr>
<tr>
<td>Control Group Radio</td>
</tr>
<tr>
<td>50%</td>
</tr>
<tr>
<td>Samples per HIT</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>HIT per iteration</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>Threshold</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>Interval</td>
</tr>
<tr>
<td>30 seconds</td>
</tr>
<tr>
<td>Reward</td>
</tr>
<tr>
<td>USD 0.01</td>
</tr>
</tbody>
</table>

5.1.2.1 Experiment results

Results of active learning with MTurk labeling are shown in Table 13.

Table 13. Experiments on digits OCR data set: results of using active learning with MTurk learning

<table>
<thead>
<tr>
<th>Amount of labeled samples</th>
<th>Control group validation accuracy</th>
<th>5-fold validation accuracy</th>
<th>Actual accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>82.67 %</td>
<td>60.33 %</td>
<td>93.62 %</td>
</tr>
<tr>
<td></td>
<td>(84.00 / 84.00 / 80.00)</td>
<td>(59.00 / 60.00 / 62.00)</td>
<td>(93.45 / 94.22 / 93.20)</td>
</tr>
<tr>
<td>300</td>
<td>88.00 %</td>
<td>71.00 %</td>
<td>95.53 %</td>
</tr>
<tr>
<td></td>
<td>(87.00 / 89.00 / 88.00)</td>
<td>(76.00 / 71.00 / 66.00)</td>
<td>(95.85 / 95.57 / 95.18)</td>
</tr>
<tr>
<td>400</td>
<td>91.33 %</td>
<td>75.00 %</td>
<td>96.44 %</td>
</tr>
<tr>
<td></td>
<td>(92.00 / 90.00 / 92.00)</td>
<td>(75.00 / 75.00 / 75.00)</td>
<td>(96.39 / 96.65 / 96.28)</td>
</tr>
</tbody>
</table>
Results of passive learning with random samples are shown in Table 14.

Table 14. Experiments on digits OCR data set: results of using passive learning with randomly samples

<table>
<thead>
<tr>
<th>Amount of labeled samples</th>
<th>Actual accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>90.38 %</td>
</tr>
<tr>
<td></td>
<td>(90.44 / 90.37 / 90.32)</td>
</tr>
<tr>
<td>300</td>
<td>92.59 %</td>
</tr>
<tr>
<td></td>
<td>(92.69 / 92.86 / 92.22)</td>
</tr>
<tr>
<td>400</td>
<td>93.87 %</td>
</tr>
<tr>
<td></td>
<td>(94.07 / 93.91 / 93.63)</td>
</tr>
<tr>
<td>500</td>
<td>94.66 %</td>
</tr>
<tr>
<td></td>
<td>(94.73 / 94.84 / 94.4)</td>
</tr>
<tr>
<td>600</td>
<td>95.20 %</td>
</tr>
<tr>
<td></td>
<td>(95.34 / 95.36 / 94.89)</td>
</tr>
<tr>
<td>800</td>
<td>96.19 %</td>
</tr>
<tr>
<td></td>
<td>(96.03 / 96.17 / 96.37)</td>
</tr>
<tr>
<td>1000</td>
<td>96.46 %</td>
</tr>
<tr>
<td></td>
<td>(96.53 / 96.37 / 96.47)</td>
</tr>
<tr>
<td>1200</td>
<td>96.62 %</td>
</tr>
<tr>
<td></td>
<td>(96.48 / 96.69 / 96.69)</td>
</tr>
<tr>
<td>1500</td>
<td>97.13 %</td>
</tr>
<tr>
<td></td>
<td>(97.06 / 97.22 / 97.1)</td>
</tr>
<tr>
<td>1700</td>
<td>97.29 %</td>
</tr>
<tr>
<td></td>
<td>(97.21 / 97.22 / 97.44)</td>
</tr>
<tr>
<td>2000</td>
<td>97.5 %</td>
</tr>
<tr>
<td></td>
<td>(97.31 / 97.63 / 97.56)</td>
</tr>
<tr>
<td>2300</td>
<td>97.72 %</td>
</tr>
<tr>
<td></td>
<td>(97.76 / 97.69 / 97.72)</td>
</tr>
</tbody>
</table>
5.1.2.2 Evaluations and discussion

5.1.2.2.1 Amount of labeled samples

The following chart shows the relationship between amount of labeled samples involved and the resulted accuracy:

![Chart 1. Experiments on digits OCR data set: relationship between accuracy and amount of labeled samples](chart.png)

It is obvious that for any target accuracy, using the active learning and MTurk labeling ability offered by the framework required much fewer labeled samples than passive learning with random samples. To achieve accuracy 97.72%, passive learning used 2300 random samples while the framework’s active learning strategy needed only 600 samples. In other words, the framework saved the effort of labeling 1700 samples. Also, when accuracy was higher than 95%, the amount of labeled samples needed for passive learning to improve accuracy increased dramatically with a high exponent \( y = 3E-17e^{0.4686x} \) while active learning offered by the framework maintained an almost linear trend line.
5.1.2.2 Validation methods accuracy

As mentioned in 4.2.4 Accuracy evaluation, because of the nature of active learning, 5-fold validation had greatly underestimated the accuracy of the classifier and thus control group validation was implemented to provide a more accurate evaluation. The following chart shows the comparison on 5-fold validation, control group validation and the actual accuracy.

**Chart 2. Experiments on digits OCR data set: comparison on validation methods**

Chart 2 shows that 5-fold validation greatly underestimated the accuracy by 20~30% as expected. Meanwhile, control group validation gave much closer values to the actual accuracy. Yet, it has still underestimated the accuracy by 2~10%. However, it is observable that with the increase in involved samples, both 5-fold validation and control group validation could give better accuracy assessment. Especially for control group validation, when the amount of labeled samples met 600, its accuracy assessment was just 2.72% lower than the actual percentage.
5.1.3 Experiment on the spam SMS data set

The spam SMS data set was obtained from UCL machine learning repository. Samples were SMS message retrieved from various sources including Grumbletext and NUS SMS Corpus. Features were the TF-IDF values extracted from samples with maximum feature equal to 100. The composition of this dataset is shown in table 15.

Table 15. Experiments on spam SMS data set: Composition data set

<table>
<thead>
<tr>
<th>Class</th>
<th>Spam</th>
<th>Not spam</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amount</td>
<td>747</td>
<td>4,827</td>
</tr>
</tbody>
</table>

Total samples: 5,574

Details on http://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection

Machine learning algorithm chosen for the spam SMS dataset was Nearest Neighbors, which is commonly adopted for text content recognition such as content based spam filters. Its detail configuration is shown in Table 16. For active learning and MTurk labeling, sampling range was set to be 40 folds of samples per iteration and control group radio was set to be 50%. Detail configuration is shown in Table 17.

Table 16. Experiments on spam SMS data set: machine learning configuration

<table>
<thead>
<tr>
<th>Major machine learning configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm</td>
</tr>
<tr>
<td>Classifier type</td>
</tr>
<tr>
<td>n_neighbors</td>
</tr>
<tr>
<td>weights</td>
</tr>
<tr>
<td>Internal algorithm</td>
</tr>
</tbody>
</table>
Table 17. Experiments on spam SMS data set: Active learning and MTurk configuration

<table>
<thead>
<tr>
<th>Active learning and MTurk configuration</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Enlarge sampling range</td>
<td>40 folds</td>
</tr>
<tr>
<td>Control Group Radio</td>
<td>50%</td>
</tr>
<tr>
<td>Samples per HIT</td>
<td>10</td>
</tr>
<tr>
<td>HIT per iteration</td>
<td>2</td>
</tr>
<tr>
<td>Threshold</td>
<td>0</td>
</tr>
<tr>
<td>Interval</td>
<td>30 seconds</td>
</tr>
<tr>
<td>Reward</td>
<td>USD 0.01</td>
</tr>
</tbody>
</table>

5.1.3.1 Experiment results

Results of active learning with MTurk labeling are shown in Table 18.

Table 18. Experiments on spam SMS data set: results of using active learning with MTurk learning

<table>
<thead>
<tr>
<th>Amount of labeled samples</th>
<th>Control group validation accuracy</th>
<th>5-fold validation accuracy</th>
<th>Actual accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>92.00 %</td>
<td>79.67 %</td>
<td>93.76 %</td>
</tr>
<tr>
<td></td>
<td>(88.00 / 93.00 / 95.00)</td>
<td>(83.00 / 78.00 / 78.00)</td>
<td>(93.74 / 94.13 / 93.41)</td>
</tr>
<tr>
<td>300</td>
<td>92.67 %</td>
<td>81.00 %</td>
<td>94.73 %</td>
</tr>
<tr>
<td></td>
<td>(90.00 / 95.00 / 93.00)</td>
<td>(81.00 / 83.00 / 79.00)</td>
<td>(94.51 / 94.87 / 94.8)</td>
</tr>
<tr>
<td>400</td>
<td>93.67 %</td>
<td>82.33 %</td>
<td>95.60 %</td>
</tr>
<tr>
<td></td>
<td>(92.00 / 95.00 / 94.00)</td>
<td>(85.00 / 85.00 / 80.00)</td>
<td>(95.62 / 95.62 / 95.55)</td>
</tr>
<tr>
<td>500</td>
<td>94.33 %</td>
<td>83.67%</td>
<td>96.30 %</td>
</tr>
<tr>
<td></td>
<td>(94.00 / 94.00 / 95.00)</td>
<td>(83.00 / 85.00 / 83.00)</td>
<td>(96.2 / 96.41 / 96.28)</td>
</tr>
<tr>
<td>600</td>
<td>94.67 %</td>
<td>83.67%</td>
<td>96.69 %</td>
</tr>
<tr>
<td></td>
<td>(96.00 / 94.00 / 94.00)</td>
<td>(83.00 / 82.00 / 86.00)</td>
<td>(96.66 / 96.64 / 96.77)</td>
</tr>
</tbody>
</table>

Results of passive learning with random samples are shown in Table 19.
Table 19. Experiments on digits OCR data set: results of using passive learning with randomly samples

<table>
<thead>
<tr>
<th>Amount of labeled samples</th>
<th>Actual accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>89.06 %</td>
</tr>
<tr>
<td></td>
<td>(89 / 89.3 / 88.89)</td>
</tr>
<tr>
<td>300</td>
<td>89.83 %</td>
</tr>
<tr>
<td></td>
<td>(90.51 / 89.45 / 89.54)</td>
</tr>
<tr>
<td>400</td>
<td>90.94 %</td>
</tr>
<tr>
<td></td>
<td>(91.17 / 90.88 / 90.78)</td>
</tr>
<tr>
<td>500</td>
<td>92.41 %</td>
</tr>
<tr>
<td></td>
<td>(92.64 / 92.57 / 92.03)</td>
</tr>
<tr>
<td>600</td>
<td>93.14 %</td>
</tr>
<tr>
<td></td>
<td>(92.48 / 93.86 / 93.07)</td>
</tr>
<tr>
<td>1000</td>
<td>95.03 %</td>
</tr>
<tr>
<td></td>
<td>(95.17 / 95.17 / 94.74)</td>
</tr>
<tr>
<td>1200</td>
<td>95.57 %</td>
</tr>
<tr>
<td></td>
<td>(95.64 / 95.55 / 95.51)</td>
</tr>
<tr>
<td>1500</td>
<td>96.31 %</td>
</tr>
<tr>
<td></td>
<td>(96.37 / 96.28 / 96.27)</td>
</tr>
<tr>
<td>1700</td>
<td>96.47 %</td>
</tr>
<tr>
<td></td>
<td>(96.49 / 96.48 / 96.43)</td>
</tr>
<tr>
<td>1900</td>
<td>96.66 %</td>
</tr>
<tr>
<td></td>
<td>(96.61 / 96.73 / 96.63)</td>
</tr>
</tbody>
</table>
5.1.3.2 Evaluations and discussion

5.1.3.3.1 Amount of labeled samples

Chart 3 shows the relationship between amount of labeled samples involved and the resulted accuracy:

![Chart 3. Experiments on spam SMS data set: relationship between accuracy and amount of labeled samples](chart)

Similar to using digits dataset with Support Vector Machine, experiment on spam SMS dataset with Nearest Neighbor showed that the framework could greatly reduce samples amount. In this experiment, active learning with MTurk used only 200 samples to achieve accuracy 93.76% which passive learning could not met with 600 random samples. However, the growth of accuracy was slower than using digits OCR data set. When active learning used 600 samples, the accuracy was 96.69%, which was lower than 97.72% the results of digits dataset. However, meeting this accuracy for passive learning needed 1900 random samples, so the framework actually still saved the effort of labeling 1300 samples. Similar to digits
dataset, the active learning and MTurk labeling ability provided by the framework could keep the increase curve close to linear growth when samples required by passive learning grew exponentially.

5.1.3.3.2 Validation methods accuracy

Chart 4 shows the comparison on 5-fold validation, control group validation and the actual accuracy in the spam SMS dataset experiments.

Chart 4. Experiments on spam SMS data set: comparison on validation methods

5-fold validation underestimated the accuracy by 23~26% while control group validation by only 1~2%. The difference between control group validation accuracy and actual accuracy maintained stable are very small. It is a good sign indicating that control group validation can accurately assess the performance of classifiers.

5.1.4 Brief summary

Experiments were carried out on two very different data set with different learning algorithms. The results show that the framework can significantly reduce the amount of labeled samples required and thus reduce the cost and effort required to obtain accuracy classifiers. Although data sets and the choice of
learning algorithms can affect the active learning efficiency, the framework can at least reduce the amount of required samples to one third of the original. Also, since each HIT had 10 samples and was paid with USD 0.01, the total cost of obtaining labels for 600 samples via MTurk using this framework is USD 0.6 only. In other words, users can obtain a highly accurate classifier by paying USD 0.6 instead of labeling over 2000 samples by themselves. This result meets the objective of developing this framework.

For the validation methods, 5-fold validation cannot provide accurate assessment on the classifier performance. This result is expected. Control group validation as a compromised option slightly underestimates the classification accuracy in general, but it is good enough for even entry-level users to evaluate the learning result and determine a suitable point of stopping further learning.
### 5.2 Effort required to obtain a classifier

#### 5.2.1 Comparison between the framework and alternatives

The following shows a comparison of the effort required from users to obtain a classifier using different approaches:

(assume that: passive learning needs 2000 samples; active learning needs 600 samples; labeling each sample needs 10 seconds; MTurk labeling for each HIT costs USD 0.001; coding each 150 lines takes 1 hour)

**Table 20. Effort comparison between the framework and alternatives**

<table>
<thead>
<tr>
<th>Items</th>
<th>Developing an passive learning application with self-labeling</th>
<th>Developing an passive learning application with MTurk labeling</th>
<th>Developing an active learning application with self-labeling</th>
<th>Developing an active learning application with MTurk labeling</th>
<th>Using ActiveCrowd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developing an application</td>
<td>Around 300 lines for the fundamental features without GUI</td>
<td>At least 700 lines for the fundamental features without GUI</td>
<td>At least 600 lines for the fundamental features without GUI</td>
<td>At least 1000 lines for the fundamental features without GUI</td>
<td>Around 1<del>10 lines for sample2Info, 1</del>5 lines for answer2Label and 1 minute for sklearn setting</td>
</tr>
<tr>
<td>Creating a MTurk properties file</td>
<td>N/A</td>
<td>At least 10 lines for basic properties</td>
<td>N/A</td>
<td>At least 10 lines for basic properties</td>
<td>1 minute for setting via webpage</td>
</tr>
<tr>
<td>Creating a MTurk properties file</td>
<td>N/A</td>
<td>At least 250 lines for creating HIT with 10 questions using selection answers</td>
<td>N/A</td>
<td>At least 250 lines for creating HIT with 10 questions using selection answers</td>
<td>2 minute for setting via webpage</td>
</tr>
<tr>
<td>Labeling time</td>
<td>333 minutes</td>
<td>N/A</td>
<td>100 minutes</td>
<td>N/A (done by MTurk)</td>
<td>N/A</td>
</tr>
<tr>
<td>Total payment</td>
<td>N/A</td>
<td>USD 2</td>
<td>N/A</td>
<td>USD 0.6</td>
<td>USD 0.6</td>
</tr>
<tr>
<td>Total coding effort</td>
<td>300 lines</td>
<td>960 lines</td>
<td>600 lines</td>
<td>1260 lines</td>
<td>2~15 lines</td>
</tr>
<tr>
<td>Total time</td>
<td>453 minutes</td>
<td>384 minutes</td>
<td>340 minutes</td>
<td>504 minutes</td>
<td>5 minutes</td>
</tr>
</tbody>
</table>
5.2.2 Brief summary
As shown in Table 20, the framework is able to reduce implementation effort significantly. To achieve the same active learning and MTurk labeling abilities, 1000 lines of python codes are expected to be implemented by users for each project, not to mention that they need to test their applications by themselves, while the framework needs only 2~15 lines from users to generate a robust project and application. Also, enabling MTurk interaction requires users to define 260 lines of XML syntax stuck to the Amazon MTurk XSD, while the framework enables users to configure MTurk interaction via the interface within just 5 minutes. Moreover, the framework is easy to use and it provides a graphical interface for adjusting and monitoring. It saves users much time on learning machine learning, Python coding and MTurk using, as well as adjusting applications and monitoring via command line.

6. Conclusion

6.1 Achievements
This project successfully provides a solution to the integration of active learning and crowdsourcing. The active learning ability implemented by the framework is proven to be able to reduce the amount of required labeled samples to only one third of the original. With the use of Amazon Mechanical Turk for auto labeling, the framework can perform fully automated machine learning at very low cost and users are freed from heavy labeling tasks. It can even accelerate the labeling process and save model training time from users.

Comparing to the traditional approaches which require user to develop tailored applications for each machine learning demand, the ActiveCrowd framework is able to generate highly customized projects and applications for users to build and export classification models through simple setting. Implementation and preparation time required are greatly reduced from days to just a few minutes.
Meanwhile, the web-based graphical user interface as a part of ActiveCrowd allows users to monitor and adjust the learning operations comprehensively. It greatly reduces the technical knowledge and learning time required to make use of machine learning so companies or individuals with limited skills and resources can make use of the framework and be benefited by machine learning technologies.

6.2 Future improvement
Currently the framework already meets the objectives of this project. However, several improvements can be made to further enhance the usability and flexibility of the framework.

First, although much research has shown that high percentage of answers gathered via MTurk has good quality, there is no guarantee that all of them are qualified. Instead of blindly trusting workers, it is better for the framework to have the ability of verifying answer quality. Potential solution includes creating redundancy and inserting verification or trap questions.

Secondly, the k-fold validation method cannot provide a usable assessment on classifier accuracy trained by active learning. Since the development time for the framework is limited, simple control group validation is currently adopted to overcome this problem. However, control group validation still slightly underestimate the actual accuracy. Also, control group samples does not have much contribution to the improvement of classification accuracy. Thus, labeling control group samples become an overhead to active learning. A better validation method, such as entropy-weighted k-fold validation, should be designed and implemented.

Finally, the framework’s active learning now relies on single classifier entropy based sampling strategy. Although this strategy works well, there is no guarantee that it is always the best strategy for different environments. More strategies should be designed and implemented so that users can select the most
appropriate one for their own applications. Potential strategies includes multi-classifier differential-entropy-based sampling strategy and feature distribution based sampling strategy.
## 7. Monthly logs

### Table 21. Monthly logs

<table>
<thead>
<tr>
<th>Month</th>
<th>Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014</td>
<td></td>
</tr>
<tr>
<td>August</td>
<td>Establish project topic and study pass research papers</td>
</tr>
<tr>
<td>September</td>
<td>Study Amazon Mechanical Turk service and interaction methods. Evaluate the use of official APIs, boto library and the command line tool</td>
</tr>
<tr>
<td></td>
<td>Study Python</td>
</tr>
<tr>
<td></td>
<td>Implement MTurk connection layer to enable interaction between Python applications and MTurk services</td>
</tr>
<tr>
<td>October</td>
<td>Add PostgreSQL support to MTurk connection layer and improve the communication methods</td>
</tr>
<tr>
<td></td>
<td>Study machine learning and scikit-learn</td>
</tr>
<tr>
<td></td>
<td>Implement learning automation layer using stream-based sampling</td>
</tr>
<tr>
<td></td>
<td>Integrate MTurk connection layer with learning automation layer</td>
</tr>
<tr>
<td>November</td>
<td>Replace stream-based sampling by pool-based sampling and improve efficiency of learning automation layer</td>
</tr>
<tr>
<td></td>
<td>Implement k-fold validation</td>
</tr>
<tr>
<td></td>
<td>Add PostgreSQL support to learning automation layer</td>
</tr>
<tr>
<td></td>
<td>Study various approaches for developing graphic user interface for Python applications</td>
</tr>
<tr>
<td></td>
<td>Complete and submit interim report I</td>
</tr>
</tbody>
</table>
| December | Graphic user interface design  
| Study and adopt bootstrap for the interface  
| Establish web-based interface structure and layout template with Flask  
| Improve database design to reduce redundancy and improve efficiency |

| 2015 | Transform flask framework structure to blueprints and models  
| Implement project dashboard layout template  
| Modify database design for the concept of projects  
| Implement framework setup page template, about page template, project creation page template, project overview page template, active learning and MTurk setting page template, scikit-learn setting page template, user-defined function page template and remove page template  
| Implement server side python script to handle request and response to framework setup page, about page, project creation page, project overview page and part of the remove page  
| Establish service side to client side feedback mechanism using session for message passing and JavaScript for message parsing  
| Implement message.js and form.js for form layout control and validation |

| January | Implement project execution page template, project log page template, project sample page template, project statistic page template and project export page template  
| Implement server side python script to handle request and response to all pages.  
| Enrich the interface content. For project execution page,
implement sample import function; For MTurk setting page, implement the ability of generating MTurk properties file and MTurk question file; For sample page, enable instant prediction on samples; For remove page, complete the sample removal function, label removal function and MTurk account clearance function; For export page, develop stream data transfer method for exporting large sample CSV file

Add support of Decision Tree and Naive Bayes by modifying the scikit-learn page and server side scripts. Generalize the learning automation layer and remove learning algorithm specific content. Create templates of client application for each learning algorithm and implement the template rendering strategy

Implement thread creation, monitoring control for client application execution via the interface. Add logging and recording abilities to learning automation layer and MTurk communication layer and enable the ability of displaying logs in project execution page instantly

Study d3.js and SVG

Implement line chart rendering, pie chart rendering and JSON data retrieve mechanism using JavaScript with d3.js for displaying project and execution statistic

Complete and submit interim report II

| March | Add more detail configuration options such as enlarge sampling radio to improve framework flexibility |
|       | Add support to Nearest Neighbors and Ensemble methods |
|       | Add control group validation method. Implement control group validation in the learning automation layer. Modify the active learning and MTurk setting page to accept control group radio parameter. Modify the line chart in statistic page to display control group validation accuracy |
|       | Implement the interface home page |
| Test the framework with handwritten digits data set and spam SMS data set comprehensively and fix all discovered errors |
| Perform experiments using handwritten digits data set and spam SMS data set to evaluate the learning performance and validation method accuracy |
| Do final report |
| April | Complete and submit the final report |
|      | Complete and submit demonstration clip |
|      | Complete and submit project screenshot |
|      | Prepare presentation |
References


Appendix I: List of charts

Chart 1. Experiments on digits OCR data set: relationship between accuracy and amount of labeled samples.................................................................87

Chart 2. Experiments on digits OCR data set: comparison on validation methods ..................................................................................................................88

Chart 3. Experiments on spam SMS data set: relationship between accuracy and amount of labeled samples.................................................................92

Chart 4. Experiments on spam SMS data set: comparison on validation methods ..................................................................................................................93
Appendix II: List of tables

Table 1. Execution environment and packages used .......................................................... 31
Table 2. The MTurk pending table design ........................................................................ 34
Table 3. The MTurk record table design .......................................................................... 35
Table 4. The MTurk failure table design ......................................................................... 36
Table 5. The MTurk answer table design ......................................................................... 36
Table 6. The sample table design .................................................................................... 50
Table 7. The execution record table design ..................................................................... 51
Table 8. The project table design .................................................................................... 59
Table 9. Supported classification models and classifier types ....................................... 63
Table 10. Experiments on digits OCR data set: Composition data set ......................... 84
Table 11. Experiments on digits OCR data set: machine learning configuration ......... 85
Table 12. Experiments on digits OCR data set: Active learning and MTurk configuration .......................................................................................................................... 85
Table 13. Experiments on digits OCR data set: results of using active learning with MTurk learning .............................................................................................................. 85
Table 14. Experiments on digits OCR data set: results of using passive learning with randomly samples ........................................................................................................ 86
Table 15. Experiments on spam SMS data set: Composition data set ....................... 89
Table 16. Experiments on spam SMS data set: machine learning configuration ........ 89
Table 17. Experiments on spam SMS data set: Active learning and MTurk configuration .......................................................................................................................... 90
Table 18. Experiments on spam SMS data set: results of using active learning with MTurk learning .............................................................................................................. 90
Table 19. Experiments on digits OCR data set: results of using passive learning with randomly samples........................................................................................................... 91

Table 20. Effort comparison between the framework and alternatives.................. 95

Table 21. Monthly logs ................................................................................................................. 99
Appendix III: List of figures

Figure 1. Differences that BigDigSci makes [9] ................................................................. 15
Figure 2. Illustration of stream-based selective sampling operation .................... 24
Figure 3. Illustration of pool-based sampling operation ............................................. 25
Figure 4. A HIT created by Amazon Inc. for recommendation validation ............ 27
Figure 5. Instructions and some answers of the experiment that asked workers to label the joints’ position. ................................................................. 28
Figure 6. Design of the framework .............................................................................. 30
Figure 7. A form for users to specify HIT properties ................................................. 39
Figure 8. Place for users to upload an existing properties file ................................. 39
Figure 9. The form for user to create a new question file......................................... 41
Figure 10. Fields for inputting regex and error message field for Free text answer with regex ......................................................................................... 42
Figure 11. An extra block for user to add new selection ........................................... 42
Figure 12. A selection True with value 1 is created. Another new selection named False is going to be added ................................................................. 42
Figure 13. Python code editor for users to implement the sample2Info function . 46
Figure 14. Python code editor for users to implement the answers2Labels function .............................................................................................................. 49
Figure 15. Page displayed after project creation ......................................................... 61
Figure 16. Page for configuring MTurk basic setting .................................................... 61
Figure 17. Page for configuring scikit-learn ................................................................. 64
Figure 18. The framework check whether the pre-requirements are fulfilled...... 65
Figure 19. Page for user the upload samples and execute client applications ... 67
Figure 20. Structure of the web application (server-side) structure .................... 70
Figure 21. Illustration of the layout relationship between layout.html and dashboardLayout.html ................................................................. 73
Figure 22. Line chart of active learning performance ........................................... 73
Figure 23. On-hover tooltip and vertical indicator .............................................. 76
Figure 24. Line chart animation timeline ........................................................ 77
Figure 25. Pie chart of duration rate ................................................................. 78
Figure 26. The skeleton of pie chart in Figure 25 .............................................. 80
Figure 27. Pie chart animation timeline ........................................................ 83