Proposal for Applying machine learning to the code interaction GSoC Project

About me

- **Name:** Prasanna Patil
- **College:** L.D. College of Engineering, Ahmedabad, Gujarat, India
- **Field of study:** B. Tech in Computer engineering
- **Email:** prasannapatil08@gmail.com
- **GitHub**: prasanna08

Project details

Project name: Applying machine learning to the code interaction

Why are you interested in working with Oppia?

I am interested in this project because it requires both frontend as well as backend development. This is good chance to learn how to create and manage a good system, which can be used in actual real world applications.

What interests you about this project? Why is it worth doing?

This project requires understanding of different machine learning techniques which can be used for classifying program code. This provides me a chance to deploy a machine learning technique into production environment of some software which is awesome experience to have.

Prior experience

I have implemented a few machine learning algorithm myself in my GitHub repo here. Therefore I am well aware of different algorithms that can be used with for classification. I have been working with Oppia and currently I am maintainer of Emails project. The project has given me a good insight of backend working of Oppia. I have worked on following issues / features:

- Implementing functionality to send bulk emails to users who qualify specified criteria.
- Implementing functionality to send feedback message emails to exploration owners.
- Implementing functionality to receive replies to feedback messages from users.
- Implement functionality to send feedback message emails to all thread participants.
- Implement Mailgun API as an alternative to GAE mail service for sending emails.

Project description

Oppia provides online learning platform for all students across the globe. It is intelligent learning platform where creators can create explorations and students learn from them in interactive way. The aim of this project is to code input interaction more intelligent by using machine learning methods and deploy it in production environment by the end of internship.

Code classification

Currently Oppia supports many ways to attach some response to code input. This includes response based on specific output or program written in code input editor. The classification will take place when code input fails to qualify any of the hard rules specified by creator.

Training data

There are 3 possible ways for creator to specify training data for any interaction.

1. Creator can give a specific input to Oppia, and then Oppia will classify that input to some answer group. If the input given by the creator is classifiable using hard rules, then it will not be considered for inclusion in the training dataset. Otherwise, the classifier will be used to classify that input. If the creator is not satisfied with the predicted answer group, then they will be asked to assign the input to some other answer group (possibly a new one). The input will then be added as training data to the selected answer group.
2. Creator can view history of answers and assign answer groups to any input in history. This input then will be added to training data of of that answer group.
3. We can cluster the answers in history and show this clusters to creator and ask him if he wants to assign this cluster to some answer group. If creator assigns some answer group to a cluster then those clustered inputs will be added to training data of that answer group.
    a. Perform clustering when this option is chosen by the creator in the frontend. This has 2 advantages:
        i. If we used one off job then deciding when to regenerate clusters is not quite clear. If we generate cluster only when creator asks them we can re generate them when creator says so.
        ii. We don't need to store clusters. We can generate them and temporarily store them somewhere. Once creator has assigned some answer group we can delete clusters.

Unsupervised clustering

- 3rd option to generate dataset is to perform unsupervised clustering and show these clusters to creator. If creator wants then he can assign the entire cluster to answer group.
- This will add all answers in that cluster to training data of that answer group.
- This requires unsupervised learning algorithm associated with that interaction. It might not be possible to have an unsupervised clustering algorithm for some interaction. Therefore 3rd option is displayed only when unsupervised algorithm for that interaction is present.
- When to generate clusters?
    - Clusters will be generated / re-generated when creator selects 3rd option for adding training data in teach oppia modal.
    - We will store clusters temporarily so that creator can assign answer group to a cluster then we can delete clusters from storage. We will generate them when creator asks and when he has added clusters to feedback group we will discard it.
- When to update training data?
    - If number of unresolved answer + answer classified using classifier is > X (X is threshold value). We will notify creator that the old clusters might be stale, please update clusters. Its creator’s choice whether to update clusters or not.
- How to know if (unresolved answer + answer classified using classifier is > X) condition is satisfied?
    - Add a new parameter in ClassifierModel which will store count of unresolved answers + answers classified using classifier.
- How to keep track of inputs in an cluster (we have to store clusters temporarily somewhere so how should we store it)?
    - We should store clusters temporarily somewhere so that we don’t have to transfer entire cluster group to the frontend. We can show several samples from a cluster and if creator assigns some answer group to that cluster we can add entire cluster group in the training set in backend by retrieving cluster from temporary storage.
- Suppose if creator has already used unsupervised cluster for generating training data and then when he uses “teach oppia using clusters” again in teachOppiaModal we will show re-generated clusters. That means that we will not store old clusters anywhere.

Training classifier

Once we have enough training data for an answer group in a state we can train a classifier for this state.

- The training process will include all answer groups which have number of training examples more than X (X being threshold value). We should have at least 2 (or more) such answer groups.
- We will train classifier on save / publish and creator has modified training data of any answer group.
    - It reduces retraining frequency. If we train whenever a new answer is added or removed will increase load on the system.
      Creator may add some answers and then consecutively remove some answers which will cause a lot of retraining of classifier (since it is possible to add answers in bulk by using unsupervised cluster but removing them is still single answer at a time).

Complete workflow

1. Creator will submit training data using one of the following methods:
    a. By submitting a new answer and assigning it to an answer group
    b. By selecting answer from answer history and assigning it to an answer group
    c. By selecting cluster of answers and assigning it to an answer group (see unsupervised clustering for more details)
2. Once training data is submitted creator will save / publish exploration.
3. This will trigger training of classifier if following condition is satisfied
    a. There are at least 2 answer groups which have more than X (threshold) answers in training data.
4. Trained classifier will be stored in ClassifierModel
5. When a new input will be given by student which does not qualify in any if the hard rules specified by creator, answer will be submitted to backend for classification using classifier. This will use classifier stored in ClassifierModel for classification.
6. Classifier will predict answer group for this answer and return it to frontend.

Performance measuring

There are several metric possible to measure performance of classification algorithm. I will be using following metric for measuring performance:

1. F1 score
2. Confusion matrix
3. Accuracy of classification

I will be comparing classifier performance with “uniform” baseline which is predicting output class (answer group) uniformly at random.

Project plan

Classification algorithm

The main focus of this project is to build a classification algorithm which is fast and accurate for code classification.

There are many classification algorithms are possible each having its own trade offs.

When performing code input there are 2 types of features we need to consider:

1. Syntactic features
2. Semantic features

Classifier:

Classifier consists of three parts. We can select best strategy in each section and then combine them together to form a classifier.

1. Feature extractor: feature extractor will extract features based on syntax and semantic of input program. Extractor will then yield a vector (a vector of real values) representation of input code.
2. Supervised classifier: This vector representation will be given to a supervised classifier. Supervised classifier will do the task of associating each input program (in vector form) to its answer group.
3. Unsupervised classifier: unsupervised classifier will take vector representation of program and generate clusters of them. It will create groups of similar programs.

The exact classification algorithm to be implemented will be decided during community bonding time. Different strategies which can be used in above sections:

- Syntactic feature extraction
    - Doc2Vec:
        - This method treats a program as a simple text document. It captures syntactic features very efficiently. Programs having similar syntax are near to each other in vector space created by

            this method. It is internally dependent on word2vec method which creates word vectors.
        - It requires its own separate network to be trained for vectorizing documents. Word2vec also requires its own separate network to be trained. So first we have to train word2vec and then doc2vec and then we can train classifier. It fails to capture semantic aspect of the program.
    - Bag of Words:
        - This is very simple method for feature extraction from text documents.
        - However it neither maintains syntactic nor semantic aspect of program.
- Semantic feature extraction:
    - Roles of Variables:
        - We can use ROV for semantic feature extraction. There are 2 ways to extract ROV features.
            - In first method, we will first identify all variables in the program. Then we decide role of each variable. We will have fixed number of roles for a variable. Then we will calculate total number of variables that have a particular role in program. For e.g. consider we have 3 roles, r1, r2 and r3, for any variable. Then we can calculate that there are 3 variables for r1, 1 variable having role r2 and 4 variable having role r3. From this information we can construct a vector (3, 1, 4). This vector is semantic feature of program. This is similar to BoW representation of text document.
            - In second method, we will utilize doc2vec method, combined with ROV, for semantic feature extraction. Doc2vec method works as follows: consider we have a

statement “i = j + 3” in our program. Assume that i has r1 role and j has r2 role. Then in doc2vec method our input will be (r1, operator) and expected output will be (r2) (this is how doc2vec method works. In input we give “k” consecutive words and we teach model to predict “k+1”th word of the sequence).

  - Graph kernel:
    - This is kernel based method which works on graphs. Basically graph kernel is a mathematical function which takes in 2 graphs and gives a real valued number in the output.
    - We can use graph kernel with control flow graph or data flow graph of the input program. This way we can extract semantic features of program using CFG or DFG.
- Classifier:
  - MLP (Multi-Layer Perceptron):
    - MLP is a universal function approximator. It is widely used for multiclass classification when large amount of data is available.
    - Sometimes it tends to overfit the dataset and it contains many hyperparameters which need to be selected carefully. Sometimes it gets stuck at a local minima and never reaches to global minimum.
  - SVM (Support vector machine):
    - SVM is a widely known because of its kernel methods and global minimum. SVM is guaranteed to always converge to best classifier line. SVM can be trained using smaller dataset as well. Due to kernel methods SVM can always find a hyperplane where all classes are linearly separable.
    - It is hard to decide what kernel to be used and each kernel has several parameters to be decided by user.

- Unsupervised clustering:
    - K Means
        - K Means is widely used clustering algorithm for unsupervised training. It takes feature vector (real valued) as input and generates K clusters. Many methods are available under K means which perform clustering for data having different characteristics.
        - It can cluster spherical clusters only. It also creates equi-sized clusters.

I will be performing experiments on above methods to find out which works best for code classification and generates good unsupervised clusters during community bonding period.

Implementation strategy

- Community bonding period
    - During this time best strategy for code classification will be researched and finalized. First of all example dataset will be generated which will be used for benchmarking and performance measuring of candidate algorithms.

- Classifier implementation (1st month)
    - Implement classification algorithm by extending BaseClassifier class. During this month supervised classifier (feature extraction + supervised classifier) will be implemented.
    - Implement data preprocessing functions for tokenizing input data, generating vocabulary any other preprocessing function which may be required by classification algorithm in classifier_services.
- Tests:
    - Add backend tests to ensure performance and accuracy of classifier on generated dataset. This is similar to how LDAStringClassifier

performance tests are written. We will have some example dataset (generated during community bonding period) stored in yaml file and we will perform accuracy and performance tests on this dataset.
    - Make sure that preprocessing functions are working as expected.

- Unsupervised clustering (2nd month)
    - Modify BaseClassifier class to include methods for unsupervised clustering. Any existing classifier shall raise “Not implemented error” for this new method.
    - Implement unsupervised classifier for code classifier. Unsupervised classifier will generate clusters based on features extracted by feature extraction algorithms.
    - Implement storage model to store unsupervised classifier temporarily. Store clusters temporarily in this model (delete this model when exploration is saved / published).
    - Add function for unsupervised clustering in classifier_services. This function will be used for generating clusters from answer history and store them in temporary storage model. This function will take exploration_id and state in the input.
    - Modify ClassifierModel to store number of unresolved answers + answers classified by classifier since last unsupervised clustering (default will be 0, we should count this value only if creator has used unsupervised clustering at least once before).
- Tests
    - Add performance test for generating clusters unsupervised manner. We can use same dataset used for measuring performance and accuracy of supervised classifier.
    - Make sure that new added field in ClassifierModel is updated and count is correct for unresolved + classifier classified answers.

    - Ensure that clusters are stored in temporary storage and this model is deleted upon publishing / saving of exploration.
    - Make sure that unsupervised clustering function in classifier_services works as expected.

- Gathering training data from user (3rd month)
    - Implement backend handler which will trigger unsupervised clustering when creator selects “teach Oppia using clusters” option in “teachOppiaModal”. This handler will then return top X (~ 50) answers of each cluster that has been formed by unsupervised clustering algorithm to the frontend.
    - Modify “teachOppiaModal” to include option for adding training data using unsupervised clustering. In this option answers will be clustered into distinct groups and creator will assign answer group directly to a cluster. All answers belonging to that cluster will be added to training data of answer group specified by creator.
    - Implement functionality which will add all answers of a cluster to training data of specified answer group. This will require adding some sort of indicator in change list of exploration to indicate addition of training data using clusters. Whenever this indicator is encountered in change list, we will add training dataset by using clusters stored in temporary storage model.
    - Trigger classifier training whenever creator saves / published (and conditions explained in “Training classifier” section are satisfied) exploration and training dataset is modified for any one of the answer groups. (This is not yet implemented. It may happen that this functionality gets implemented outside this project by someone else working on answer classification)

- Tests:
    - Add backend tests to ensure that classifier is retrained whenever training data in an exploration is updated and creator saves / publishes exploration.
    - Make sure unsupervised clustering is triggered whenever creator selects 3rd option in “teachOppiaModal”. Make sure that backend handler performing this operation works as expected.
    - Make sure that whenever creator selects to associate a cluster with some answer group, all answers on that cluster are added to training data of that answer group (they should be removed from training data of existing answer group before adding them to new answer group).

Summer plan

Timezone

- During my internship I will be working from India (GMT + 5:30).

Working hours

- June: 7 - 8 hours on weekdays, 2 - 3 hours on weekends.
- July: 7 - 8 for first couple weeks then 5 - 6 hours on weekdays, 2 - 3 hours on weekends.
- August: 4 - 5 hours on weekdays, 2 - 3 hours on weekends.

Vacation plan

- I am out of the town from 20 May, 2017 to 25 May, 2017. I may not be able to implement anything during this time. However I will be able to communicate over the email during this time. I may be able to help out with other things during this time.

Classes

- I have classes for 4 - 5 hours per day on weekends. Therefore I will be less active on weekends.

Communication

- Communication with mentor thrice a week (flexible).
- Gitter, GTalk, Email whichever is preferable.