

TABLE OF CONTENTS:

Introduction:	2
Classification:	2
GOAL:	2
Random Forest Classifier:	3
SVM (Support Vector Machine):	4
Sequence Diagram [Working overview]:	5
Implementation:	5
Data Model:	6
Workflow:	6
a) Create Classification Model [Steps 1-5 Fig-1]	6
b) Task List View [Steps 6-8 Fig-1]	10
c) Classify a task [Steps 9-13 Fig-1]	10

Introduction:

Collaborative labeling is a process of knowing the impressions about a particular topic among the people and getting away with the baffling bias and opinions. This empowers analyzing the data and helps in forming deeper connections in social content. This project provides a platform to help with collaborative labeling and the goal is to offer people around the world to share the labeling tasks. After the labeling is done, the work is evaluated by different algorithms and generates reports online for further analysis.

Classification:

The problem of identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known.

GOAL:

Once the labelers label a task, we observe the majority votes given for the task. We now train the model with the collected results and try to predict the outcome of new posts in future and see how close we are getting towards the majority opinion. This method comes under supervised machine learning.

In order to achieve this goal, we use two classifiers to train the model and predict the outcome of the unlabelled posts. This classifier we use makes sure that the model does not overfit with the training data set.

Random Forest Classifier:

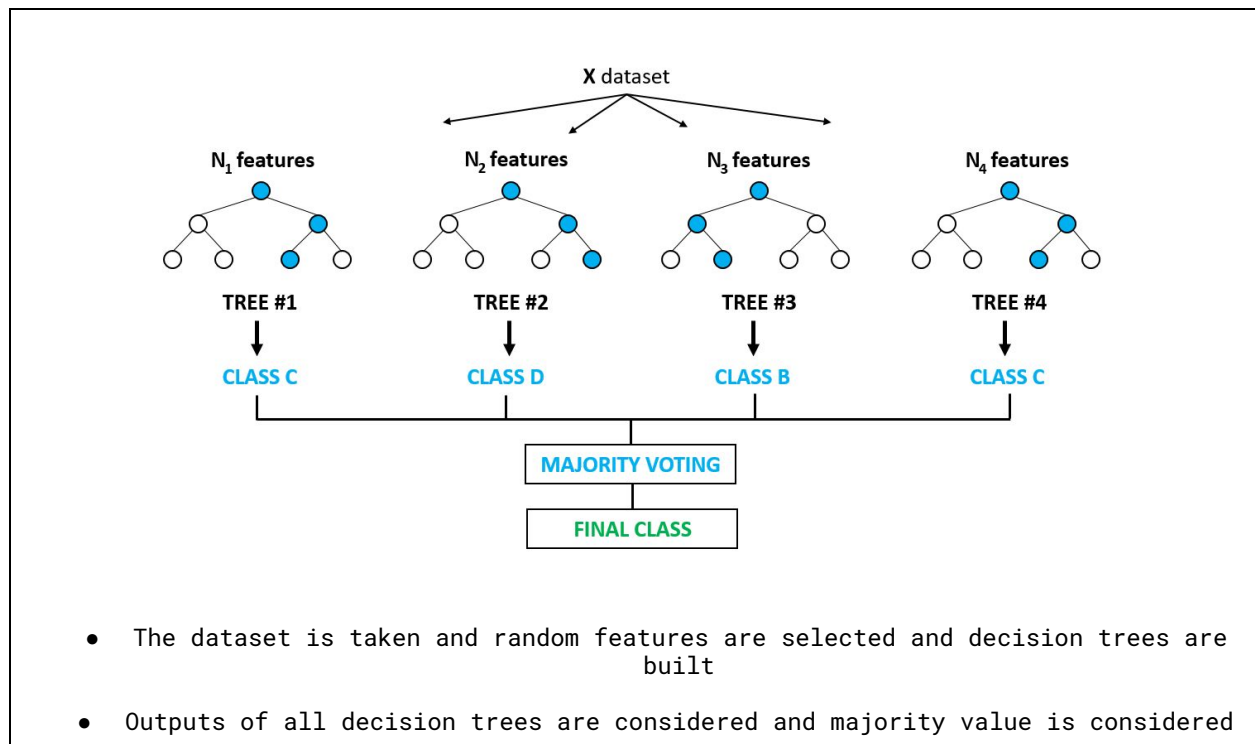
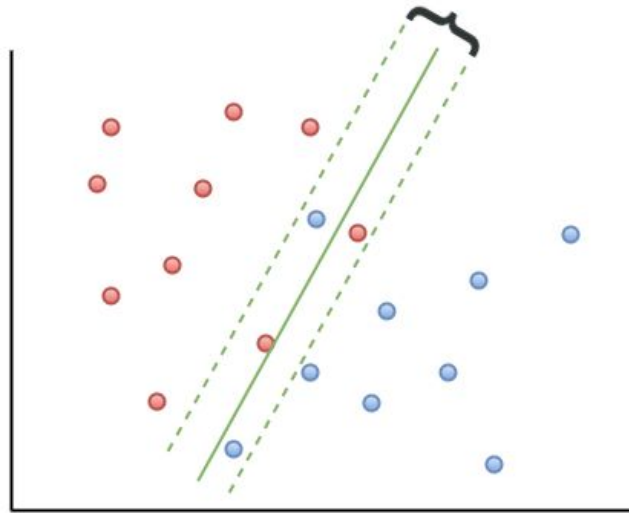


Table 1: An overview of Random forest classifier

We implement this feature in the crowdtagger application by making use of python libraries. From the library `sklearn.ensemble` we import the `RandomForestClassifier` class and build the classifier by passing it with all the required parameters. There are a lot of options to change the default options of the parameters passed to the classifier function. For example, we set the maximum depth of the decision tree, maximum number of features a sample decision tree can consider, maximum number of leaf nodes etc. We give this option to the admin who is building the classifier to change the options passed to the classifier.

SVM (Support Vector Machine):



- Support vector machines work by moving the data into relatively high dimensional space
- Finds a relatively high dimensional support vector classifier that can effectively classify the observations

Table 2: An overview of Support vector machine classifier

From the library `sklearn.svm` we import the `SVC` class and build the classifier by passing it with the parameters defined specific to this library. In SVM we particularly have some options to set like kernel, decision function shape, maximum iterations.

Finally, we call the `classifier.fit` function to train the model and build the model file. Now, to save the model, we make use of `scikit-learn` which uses python's built in persistence model `pickle`. The `pickle` module implements a fundamental, but powerful algorithm for serializing and de-serializing a Python object structure.

In the pickling process, a Python object hierarchy is converted into a byte stream and unpickling is the inverse operation whereby a byte stream is converted back into an object hierarchy. These are alternatively known as serialization or marshalling etc.

Sequence Diagram [Working overview]:

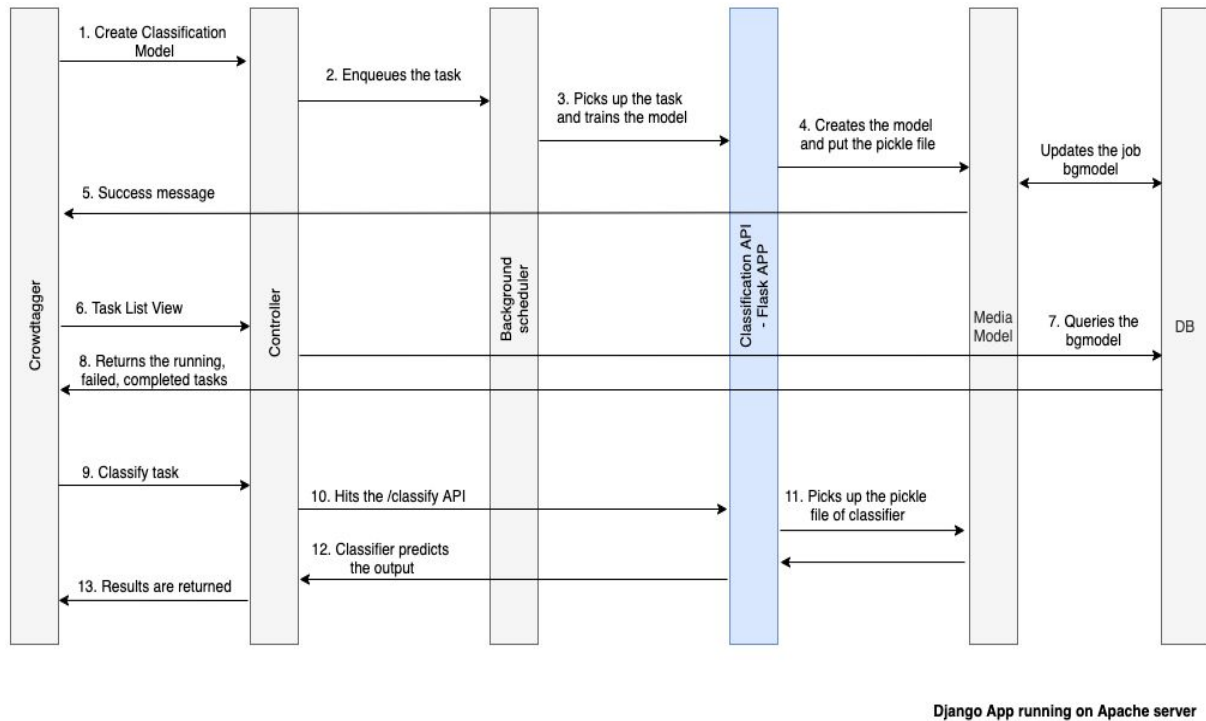


Figure 1: A sequence diagram showing overall interaction with all the components

Implementation:

The project is built using the Django framework and hosted on an apache server on dlab-rack30 server on Bolt cluster. Django is a python based free and open source web framework. This framework follows a model-template-view architectural pattern.

Functions	View	Model
Create Classifier	CreateClassificationModelView	Classify
Classifier list	ClassifyTaskListView	Classify
Delete Classifier	DeleteClassifyView	bgmodel_c.CompletedTask
Classify Data	ClassifyDataView	Classify
Success message	ClassificationModelSuccessView	N/A

	ClassifySuccessView	
--	---------------------	--

Table 3: An overview of functions and corresponding view functions and model

Data Model:

BACKGROUND TASKS (1.2.0)		
Completed tasks	+ Add	Change
Tasks	+ Add	Change
CLASSIFY		
Classifications	+ Add	Change

Figure 2: An data model used in the building the classification part on the portal

Classification
<ul style="list-style-type: none"> • Name • Description • Vectorizer • Classifier • Configuration • Task

Table 4: Fields in the classification table

Configuration is where the different options for building different classifiers are stored in json format.

Workflow:

a) Create Classification Model [Steps 1-5 Fig-1]

For using the above machine learning libraries and building the classifier model, a separate app has been built. This app is built using FLASK - a micro web framework written in python (coloured as blue in Fig-1). This app hosts 2 APIs used by the django application for building the model.

- **/train** - Builds a model by training a classifier
- **/classify** - Classifies a test data using selected classifier model built beforehand

The Flask App:

For running the flask app locally, the following commands are used.

```
export FLASK_APP = flask_app_file.py  
python3.5 -m flask run
```

Since the supported python version is 3.5 on the server, python3.5 is used to run any command. When we run the above command, it starts a server listening on ip address 127.0.0.1 and port 5000.

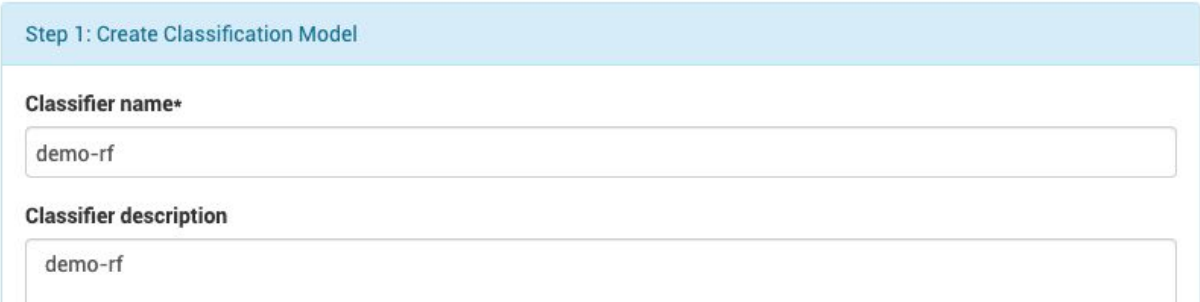
Since we want this to be running all the time, every time the server reboots, the FLASK_APP export is done using a .bashrc file and the following command to run the flask app is written in a task.sh file which executes on every reboot using crontab.

Background scheduler:

Django background task is a database backed work queue for Django.

- Creating the task functions and registering them with the scheduler.
- Setting up cron tasks (or long running process) to execute the tasks.

The background scheduler just like the flask app above gets started on every system reboot and runs in the background. The following are the sequence of steps followed in creating a classifier.



The screenshot shows a web interface for creating a classification model. It has a light blue header with the text 'Step 1: Create Classification Model'. Below the header, there are two form fields. The first field is labeled 'Classifier name*' and contains the text 'demo-rf'. The second field is labeled 'Classifier description' and also contains the text 'demo-rf'.

Figure 3: Filling out classifier name and description

Step 2: Source of Training Data

☒ CSV File

Upload task posts and labels

train (2).csv

Note : Upload a csv file with [this sample format](#)
[Posts on the first column. Labels on the last column.]

☐ Labeling Task

Figure 4: Giving the source of training data- could be new or already labeled task

Step 3: Configuration

Vectorizer*

count

Method to translate post into a vector

Classifier*

rf

Type of classifier to use. If cnn is choosen, the vectorizer will be ignored and word_embedding will be used for vectorization

Configurations for random forest classifier

Max depth

None

Figure 5: Filling out the required configurations on choosing a specific classifier

Once we fill out all the details and click the submit button, the task of creating a classifier is moved to a queue, which gets picked up the Django background scheduler process running in the background.

Task List

Runing Task:

task_name	status	run_at	complete_at	attempts	last_error	delete
demo-rf	scheduled	March 8, 2020, 8:47 p.m.	None	0		

Figure 6: Task added to the queue waiting to be executed

Task List

Runing Task:

task_name	status	run_at	complete_at	attempts	last_error	delete
demo-rf	running	March 8, 2020, 8:47 p.m.	None	0		

Figure 7: Task picked up by background scheduler

The background scheduler picks up the task 20 seconds after the task is added to the queue.

Task List

Completed Task:


task_name	status	run_at	complete_at	attempts	last_error	delete
demo-rf	success	March 8, 2020, 8:47 p.m.	March 8, 2020, 8:47 p.m.	1		

Figure 8: Successful completed task

Once the background task is completed, the task gets recorded in the Completed tasks table. As we see in the sequence diagram, the successfully completed classifier would have created a pickle file to store the model. A new directory with the name of the user_id is created and a directory with the name of classifier is created within this directory where the pickle file is stored.

```
[asunk001@dblab-rack30 demo-rf]$ ls
classify.pkl
[asunk001@dblab-rack30 demo-rf]$ pwd
/var/www/media/models/asunk001@ucr.edu/demo-rf
```

Figure 9: Created classifier model file

b) Task List View [Steps 6-8 Fig-1]

We can see all the completed tasks on the portal. The failed ones are presented in red colour as shown in the figure below and the successful ones are represented as green colour. The error message for unsuccessful tasks is also shown to be able to debug the error. The background tasks are executed in such a way that it attempts maximum 3 times before marking it as failed unless it is successful in any of the attempts.

Task List

Completed Task:



task_name	status	run_at	complete_at	attempts	last_error	delete
CNNTTEST	failed	Feb. 25, 2020, 9:27 p.m.	Feb. 25, 2020, 9:27 p.m.	3	Traceback (most recent call last): File "/usr/lib/python3.5/site-packages/background_task/tasks.py", line 43, in bg_runner func(*args, **kwargs) File "/var/www/labelingsystem/classify/tasks.py", line 10, in query_api raise Exception(response.text) Exception: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN"> <title>500 Internal Server Error</title> <h1>Internal Server Error</h1> <p>The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.</p>	
RFTEST	success	Feb. 26, 2020, 12:43 p.m.	Feb. 26, 2020, 12:43 p.m.	1		

Figure 10: List of all the tasks completed (Failed/successful)

c) Classify a task [Steps 9-13 Fig-1]

Step 1: Choose Your Classifier

Classifier*

demo-rf

Figure 11: Selecting the classifier we built earlier

Step 2: Source of Classify Data

Upload posts*

Choose File

No file chosen

Note : Upload a csv file with [this sample format](#)[Posts on the first column.] for prediction or [this sample format](#) for evaluation.

Figure 12: Selecting the source of classify data

Once we choose the classifier and source of classify data, it takes up the pickle file from the model and calls the **/classify** API and tries to predict the outcome of the posts.

Hi, asunk001@ucr.edu! Here is your classification report:

PREDICTION RESULTS:

posts	prediction label
Comprehensive up-to-date news coverage, aggregated from sources all over the world by Google News.	label1
You could measure the decline of Fox News by the drop in the quality of guests waiting in the green room.	label2
View the latest news and breaking news today for U.S., world, weather, entertainment, politics and health at CNN.com.	label1
You could measure the decline of Fox News by the drop in the quality of guests waiting in the green room.	label2
You could measure the decline of Fox News by the drop in the quality of guests waiting in the green room.	label2
...	...

Figure 13: Prediction results

We can see the model has predicted labels for the posts.