CPSC 6300

Applied Data Science

Fall 2023

Course Project: Final Project Submission
Detecting working patterns from online workers and predicting task completion.

Students:
STEPHEN BECKER, DAVID CROFT, ZACHARY TRABOOKIS
School of Computing
College of Engineering, Computing and Applied Sciences
Clemson University

Instructor:
DR. NINA HUBIG
School of Computing
College of Engineering, Computing and Applied Sciences
Clemson University

December 7, 2023

1 Introduction

The rise of AI technologies has created a vast amount of work for independent workers. These jobs include activities such as image, text, and audio annotation, as well as audio transcription, completing surveys, and many other activities. The crowdsourcing platforms (i.e., Amazon Mechanical Turk (MTurk), Yandex Toloka, etc.) offer microtasks to online workers from all over the world. This type of work is considered invisible work since it is not regulated, and few people know about it. The working patterns of these workers can be studied by analyzing telemetry data from workers completing tasks. This project aims to understand the working patterns of these workers and build AI models able to predict the activities they will take and finish. In particular, we are looking to use a machine learning model to help a crowd worker determine whether or not they should take a task based on if they would complete or not complete an individual task based on previous information and data training.

1.1 Motivation

Crowd workers spend additional time outside just completing the project task, which is not accounted for in the original requesters estimated labor cost and duration, and can be identified as invisible work. The predicted completed task output from our model can be used by a crowd worker when deciding whether to take on a new task or not. The completed task metric could be provided on a project task dashboard for the crowd worker to review prior to selecting their next task to start. This metric along with other invisible labor metrics, not identified by this paper, can be used to alter the requester task award costs or estimate time to complete for a particular task. Its important that the crowd worker gets fair compensation for the additional unpaid invisible labor work when completing a requester task. Its important to understand that evaluation of this invisible work could help change the wage for the crowd workers or provide a fair compensation estimate for a new task that a requester may create on the crowdsourcing platform.

1.2 Dataset

A research article titled *Quantifying the Invisible Labor in Crowd Work*. [TSS21] provided the original telemetry dataset which came from extending the Turkenator Chrome Extension (plugin) to collect information necessary to report on working and invisible labor events. [Gig23] The plugin helps (1) detect when a worker is doing invisible labor or when they are doing paid work; and (2) measures how much time a worker invests in each of these two activities. [TSS21, p. 4]

The plugin includes a "background process that detects the different browser events that happen on MTurk (e.g., that the worker visited another page on MTurk, or that she started typing, or began a new HIT). The page crawler detects the current MTurk domain page that the worker is on, as well as the status of the page (e.g., that the page is loaded, active, inactive, or closed). The background process focuses on detecting the human intelligence tasks (HITs) that the worker is currently doing and identifying which she has finished. In order to accomplish this, the background process polls workers task queues on MTurk every 30 seconds. From the task queue, the background process obtains the metadata and status of all the HITs the worker has accepted to do. Notice that the page crawler is the primary element that we use to detect whether the worker is completing paid labor or invisible labor. The background process helps our plugin to be able to better detect when the worker is completing HITs." [TSS21, p. 5]

The datasets that we are examining are from Amazon Mechanical Turk (MTurk) and Toloka Yandex. They observe individuals and the actions they perform while working tasks, storing them as individual events, on their respective platforms. Combined they consist of

302 unique individuals contributing 12,401,997 individual actions/events towards this dataset. There is over 4 days worth of information in the MTurk dataset; whereas, Toloka has over 43 days worth of information. Both datasets have 10 columns worth of quantitative and numerical data which includes event ids, page urls, page actions, API calls, task status, epoch timestamp, working status, person id, and two other columns that contain unusable data. Our next step was to condense our dataset and to make new columns that contained more relevant data to our problem, especially since the API calls had a bunch of useful information in the API json request.

1.3 List of Variables

The dataset did not include column names and no description was provided. After locating the research paper and talking with Carlos Toxtli through email, we were able to identify these names and descriptions. If we didn't have this initial paper it would have been hard for us to figure out exactly what was going on with the dataset provided.

- **c1 (id):** continuous id of the event generated by the Chrome Extension. [Gig23]
- **c2 (current):** url, site visited by the worker (while working)
- **c3 (event):** categorical, 18 values
  - The web browser plugin recorded multiple events, the most relevant is `PAGE_LOAD`, other events can provide repetitive information.

```
(['PAGE_LOAD', 'PAGE_BLUR', 'TAB_CHANGE', 'PAGE_FOCUS', 'PAGE_CLICK', '
    PAGE_SCROLL', 'PAGE_LAST', 'PAGE_CLOSE', 'INTERNALURL', 'PAGE_KEY', '
    PAGE_INACTIVITY', 'TAB_CLOSED', 'EXTERNALURL', 'PAGE_REACTIVATE', '
    SYSTEM_DISABLED_WORKING', 'SYSTEM_ENABLED_WORKING', 'SYSTEM_ENABLED',
    'SYSTEM_DISABLED'])
```

Listing 1: Chrome Plugin: 'events' value.

- **c4 (extra):** JSON object may include NaN values
  It provides the specifics of the tasks, it only has values for certain events. The following is a list of values used in the final cleaning csv.
  - **task_id:** continuous id of the task that the worker accepts to work on.
  - **requester_name company:** name who created the task to be completed by the worker.
  - **assignment_duration_in_seconds:** number or seconds estimated to complete the project.
  - **project.assignable_hits_count:** number of human intelligence tasks (HITs) the worker must complete for the project before they can claim completion.
  - **project.monetary_reward.amount_in_dollars:** US dollar award amount after the project has been completed by the worker.
- **c5 (platform):** categorial, 5 values The work platform in which the worker was working on (it is usually constant). Because we were focused on reading AWS Mechanical Turk data this value stated `MTURK`.

```
['OTHER', 'MTURK', 'FIVERR', 'UPWORK', 'FREELANCER']
```

Listing 2: Chrome Plugin: 'platform' value.

- **c6 (skip):** categorial, 2 values (0, 1) may include NaN value
  After talking with Carlos Toxtli, he mentioned that he didn't remember the purpose of this field. This field was unused.
- **c7 (subtype):** categorial, 29 values
  - It defines if a worker is listing the tasks available, if a task just started, or if a task was completed (submitted)
  - `FINISHED_TASK == TASK_SUBMITED` both refers to task completed (submitted is when it was recently submitted and finished when the next URL was loaded)
  - `TASK_RETURNED` When the worker decided not to work on a task.

```
['OTHER', 'TASK_STARTED', 'ADDED_TASK', 'TASK_SUBMITED', 'FINISHED_TASK', '
    TASKS_LIST', 'WORKER_DASHBOARD', 'UNKNOWN', 'TASK_FRAME', 'TASK_PREVIEW',
    'TASK_INFO', 'TASK_RETURNED', 'PLATFORM_LOGIN', 'TASK_QUEUE', 'TASK_SKIP',
     'WORKER_EARNINGS_DETAILS', 'TASK_TIMEOUT', 'WORKER_EARNINGS', '
    WORKER_QUALIFICATIONS', 'TASKS_PER_REQUESTER', 'MESSAGES_SEND', '
    TASKS_LIST_FILTER', 'WORKER_QUALIFICATIONS_PENDING', 'TASKS_PREVIEW', '
    PLATFORM_HELP', 'TASKS_PROJECTS', 'TASKS_DETAILS', 'MESSAGES_READ', '
    TASKS_APPLY']
```

Listing 3: Chrome Plugin: 'subtype' value.

- **c8 (time):** continuous, datetime (milliseconds), 1970 start date Unix Time (Week, Month, Day, Hours, Minutes, Seconds).

  This is a timestamp in milliseconds (Example: `1588994235725`). We had to convert to a date-time format with day, month, year, hour, minute, second. This datetime value represents when the Chrome Extension [Gig23] event log was occurred (Example `2020-05-09T23:31:31.151000`).
- **c9 (type):** categorial, 10 values
  It identifies if at that time the worker was working, communicating by sending messages in the platform, searching for tasks, visiting their profile. There are other types that mean that the workers changed the CONFIG of the Chrome Extension [Gig23] or the API of Toloka retrieved new tasks.

  The research paper mentioned that

```
['OTHER', 'WORKING', 'LOGS', 'SEARCHING', 'PROFILE', 'UNKNOWN', 'REJECTED', '
    COMMUNICATION', 'LEARNING', 'PROPOSAL']
```

Listing 4: Chrome Plugin: 'type' value.

- **c10 (user):** categorical, 120 values unique
  The worker id performing the crowdwork task.
- **c11:** Not relevant (an activity was taken after being recommended)
- **c12:** Not used

2 Summary of Exploratory Data Analysis (EDA)

2.1 Data Processing and Cleaning

The original MTurk dataset was generated from a Turkenator Chrome Extension (plugin) that was modified to collect information necessary to report on working and invisible labor events. We noticed that the events included `(user_id, task_id)` fields that we could leverage to group events together for our final feature set; designated values `(ADDED_TASK, FINISHED_TASK)` from a subtype event field provided ranges for start/stop for a given task. We had to filter out tasks that did not include both an `(ADDED_TASK, FINISHED_TASK)` entries as this would show incomplete information. An event timecode field was provided in milliseconds which needed to be converted to a datetime format to easily identify when an event occurred; this field was important for sorting events in a chronological order of occurrence and was necessary when aggregating the duration and count totals correctly. Flattening of an event details JSON field was needed to retrieve additional data fields `(task_id, requester_name_company, assignment_duration_seconds, project.assignable_hit_counts, project.monetary_reward_in_dollars)`. We calculated new fields for our feature set to include total, working, and invisible labor count and duration in seconds; adjusted monetary reward in dollars from the working + invisible labor; completed_task (yes/no). Processing and cleaning this data was not easy because it relied upon internal knowledge about how this data was collected through the Chrome plugin, and also figuring out how to best group individual event recordings for a task was a challenge because we had to figure out when they started and stopped.

$$\textbf{Task Pay Rate (dollars)} = \frac{\text{Task Monetary Reward (dollars)}}{\text{Task Estimate Duration (seconds)}}$$

$$\begin{aligned}\textbf{Adjusted Monetary Reward (dollars)} = {} & \text{Task Monetary Reward (dollars)} \\ & + \text{Task Pay Rate (dollars)} \times \\ & (\text{Working Labor Duration (seconds)} + \text{Invisible Labor Duration (seconds)})\end{aligned}$$

2.2 Data Visualization & Key Predictors

The projected model that will use this dataset will be constructed to predict the completion status of a task, given selected features associated with a task. As described before, completion status is a binary field, where 1 corresponds to a completed task, and 0 corresponds to a task that was never finished. As such, a classification model will be used to determine this discrete outcome with two options. Figure 1 gives a glance at the distribution of completed tasks for a subset of our cleaned, task-grouped dataset.

Due to lack of familiarity with this type of dataset, an EDA had to be performed to understand how the given features correspond with our outcome, whether a task was completed or not. Some of the features were seemingly too irregular or unrelated to task completion to use in our model, like the total time allotted by a requester for the task. However, there were three very promising features that were selected as key predictors because they showed an interesting relationship when plotted against each other for classification: monetary reward for the completion of the task, number of human intelligence subtasks ("HITs") for the task, and total number of events recorded which includes both invisible labor events and visible labor events. Figure 4 has several plots showing how these key predictors relate to one another, with orange and blue legends corresponding to whether

the task was completed or not, respectively. Figure 2 and 3 show how the task monetary reward relates to the task completion status.

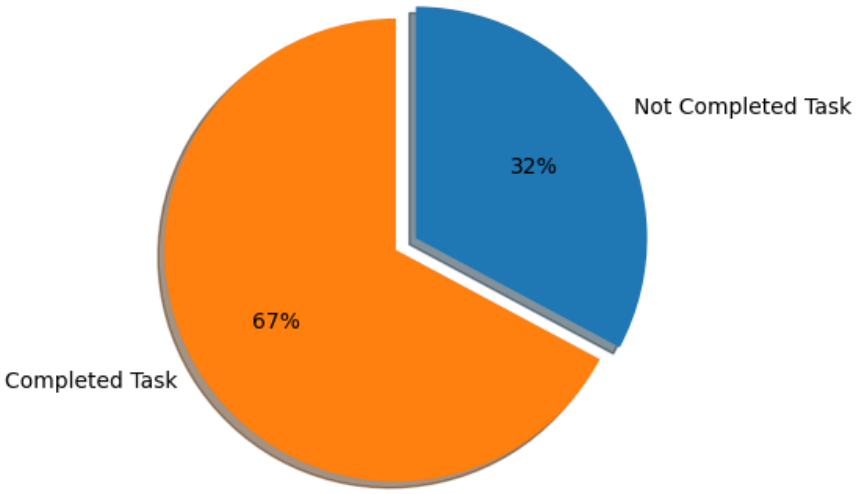

Figure 1: Distribution of Completed Task Status values.

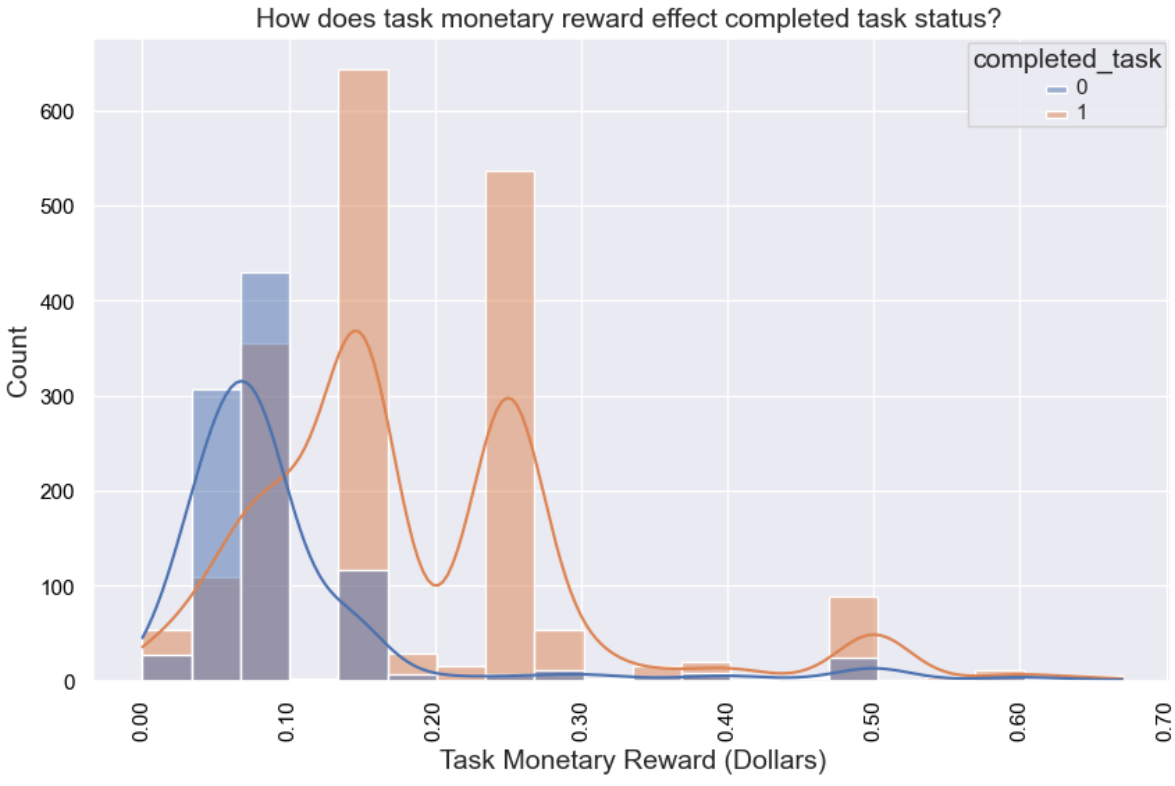

Figure 2: Histogram - Count of Task Monetary Reward for Task Completion Status.

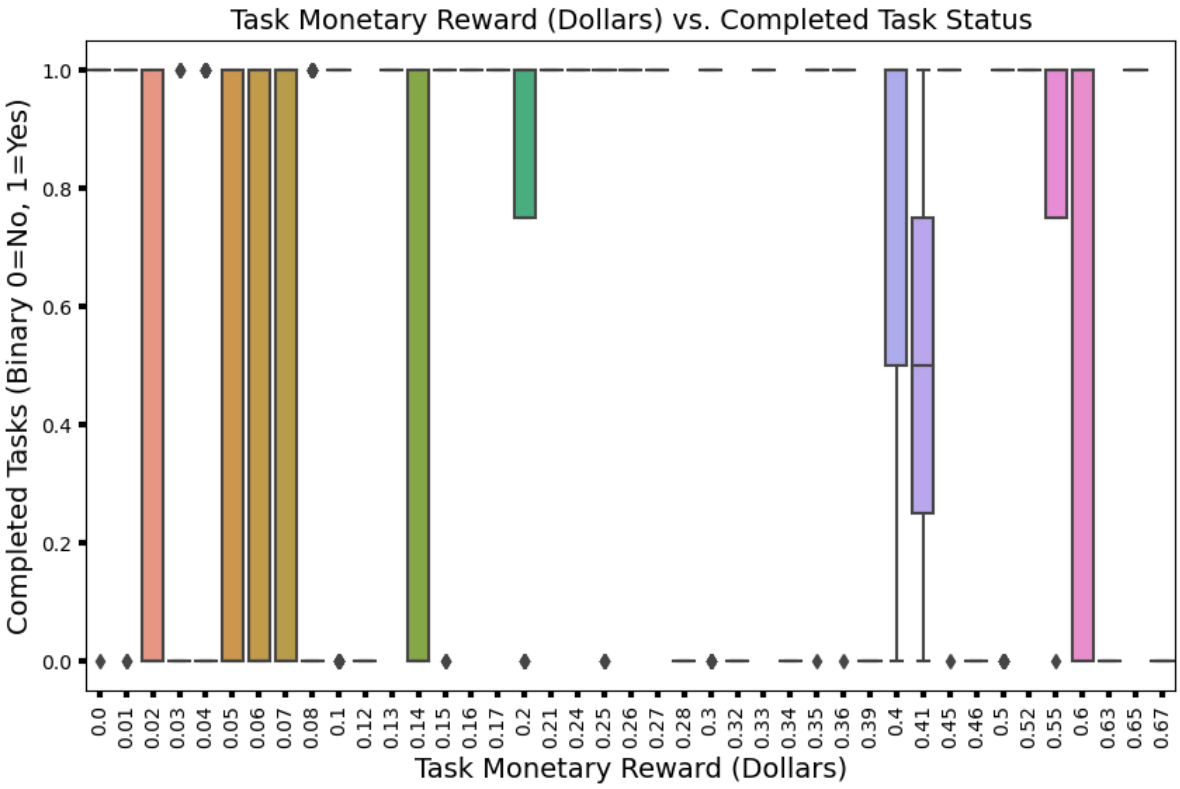

Figure 3: Box Plot - Task Monetary Reward for Task Completion Status.

3 Model Choice & Justification

The response variable for our model to predict is the binary completion status of a task accepted by a crowdsource worker: 1 for a task that is completed, 0 for a task that was not completed. To predict this binary outcome, we must select a classification model. We chose to look into k-Nearest Neighbors (kNN) and Support Vector Machine (SVM) models for this prediction output.

3.1 K-Nearest Neighbors (kNN)

This model was chosen over a linear regression classification model because, in our EDA, we found that much of our data was tightly clustered around lower values, and was not easily linear separable. The initial model has a k-value of k = 3 to avoid any erroneous classifications that fall outside the tight cluster groups in the plot of key predictors.

3.1.1 Model Analysis

For the binary classification model, a confusion matrix is an appropriate visualization to demonstrate how the model performs on test data. Figure 5 demonstrates how the model performed on a test subset of the data using a matrix, where correctly classified predictions are in the top left and bottom right squares, and misclassified predictions are in the other squares. A confusion matrix may be supplemented with classification metrics to understand the rate of model error: accuracy, the percentage of all correct classifications, precision, the percentage of correct positive classifications, and recall, the percentage of positive classifications identified by the model.

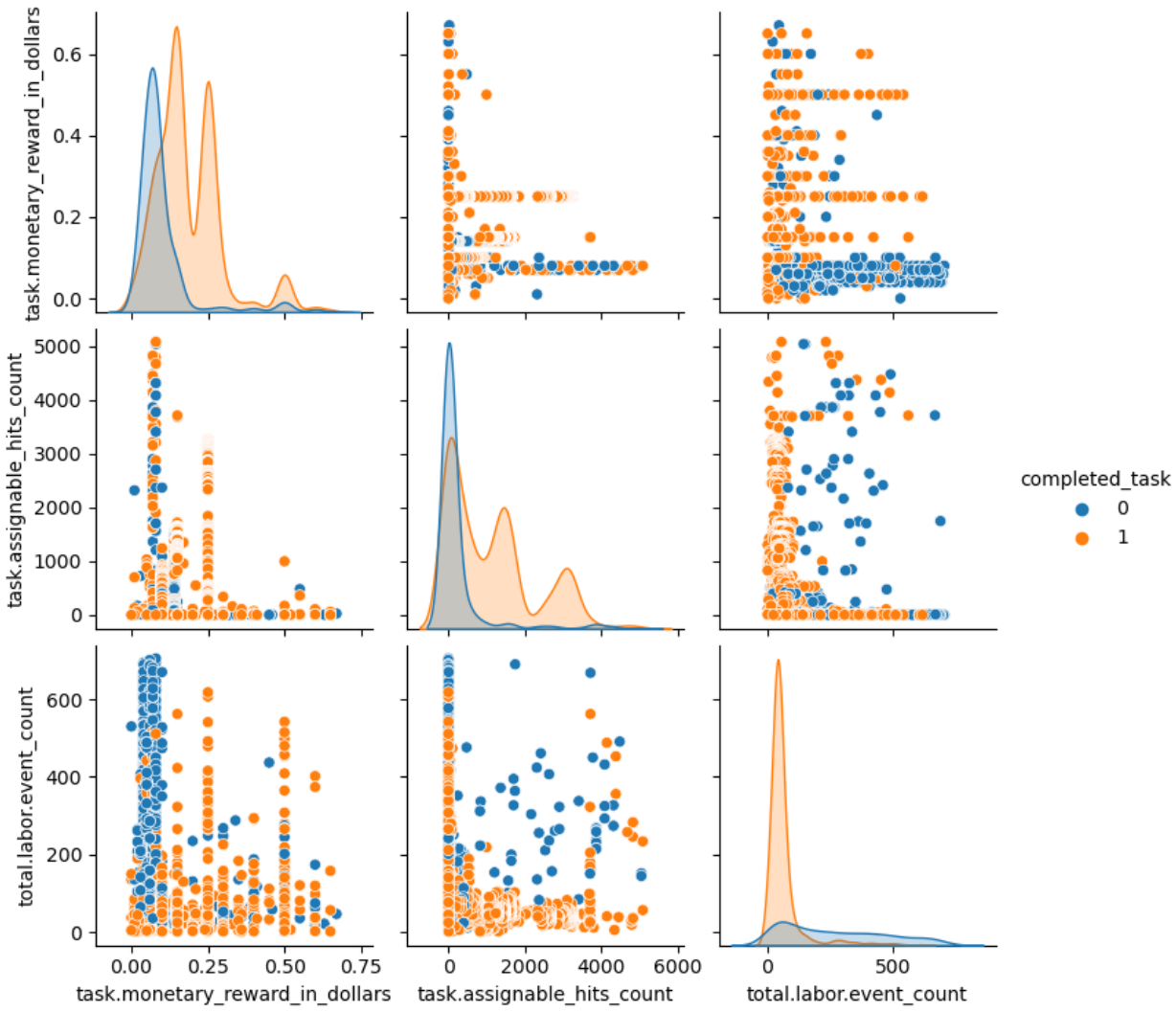

Figure 4: Seaborn Pairplot for model predictors.

3.1.2 Model Evaluation

Table 1 shows the accuracy, precision, and recall of the model, where the closer the value is to one, the less error the model had in predicting task completion. This means of error rate evaluation serves a binary classifier the best because it puts the rate of successful distinction between two choices into perspective, with respect to different measures of success. For example, if our goal of error analysis is to simply evaluate the rate of any successful classifications, accuracy serves best, but if we are most concerned with correctly identifying positive cases, we should look at recall.

Evaluation Metric	**Value**
Accuracy	0.9
Precision	0.8311688311688312
Recall	0.8797250859106529

Table 1: kNN Evaluation Metrics

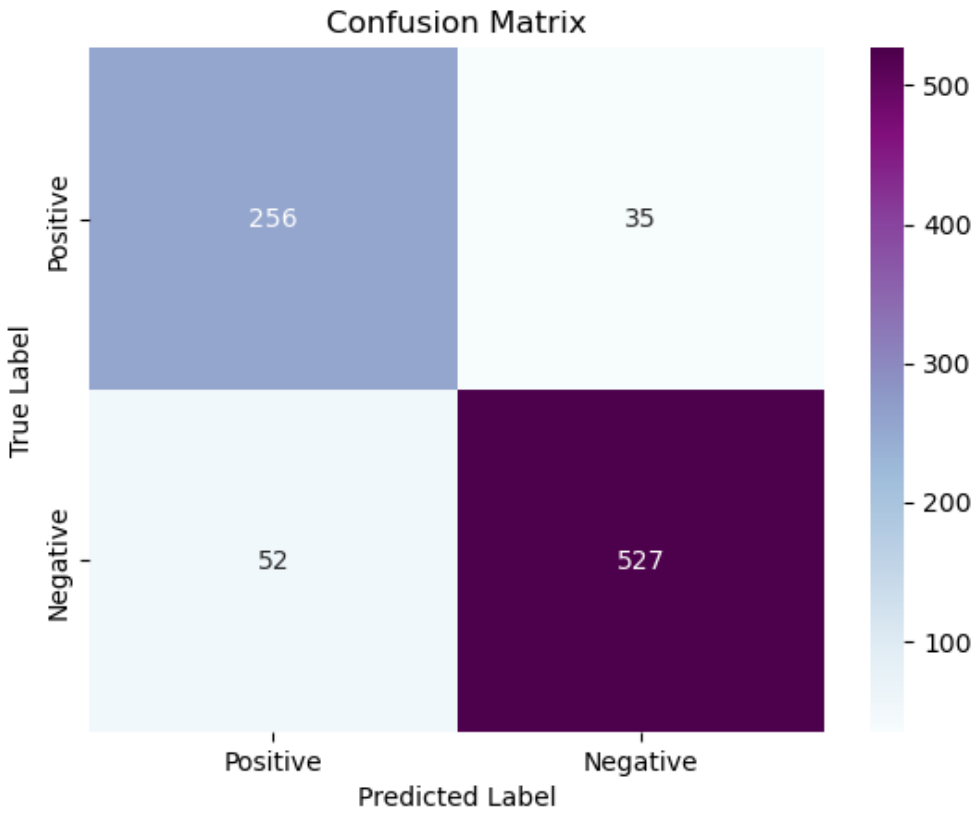

Figure 5: Confusion Matrix: K-Nearest Neighbors.

Based on the classification metrics obtained with testing data, 90% of the time our kNN model appears to correctly identify whether or not a task will be completed. Relatively speaking, this is a pretty good chance of identification, given the confusing, clustered nature of the data shown in the EDA. The lowest classification metric is precision, which means that the model has the highest rate of error when it makes a positive classification prediction. Nonetheless, this is still greater than 80%, which is still significant, given the density of the plots of the dataset's features. Overall, for an initial model, it does a fairly adequate job of fitting the data and correctly classifying unseen data.

3.2 Support Vector Machine (SVM)

The first model chosen for classification was a K-Nearest Neighbors model, which is a relatively simple method where an observation is classified based on the category of close data points in the training set. For our second model, we decided to go with a support vector machine for classification, which attempts to classify data by splitting and spacing them along a decision boundary. We chose this model because it offered a more nuanced approach to classifying a dense dataset, and the high dimensionality of our inputs would not be a problem.

3.2.1 Model Analysis

The model includes different types of kernels (algorithms that determine the similarity between input vectors). We chose a rbf or Radial Basis Function (RBF) kernel because the input data had

more clustered points from our EDA and was non-linear. We left the hyperparameter C, which controls the trade-off between the insensitive and sensitive loss, at the default of 1.0 value. The smaller the C means that the model will be more lenient in allowing larger errors and larger value means that the model will minimize the insensitive loss more. Table 2 shows the score (mean accuracy), accuracy, precision, recall, and F1-Score (harmonic mean of precision and recall) of the SVM model, where the closer the value is to one, the less error the model had in predicting task completion. As shown in Table 2, our choice of an rbf kernel is validated since the SVM gives the best accuracy, precision, recall, and F-1 score of all kernels.

Kernel	**Evaluation Metric**	**Value**
linear	Test data score	0.7310344827586207
	Accuracy	0.8367816091954023
	Precision	0.8387602385762796
	Recall	0.8367816091954023
	F1-Score	0.8290497749406808
sigmoid	Test data score	0.2574712643678161
	Accuracy	0.7977011494252874
	Precision	0.7973612554090772
	Recall	0.7977011494252874
	F1-Score	0.7975267538644472
rbf	Test data score	0.6540229885057471
	Accuracy	0.8793103448275862
	Precision	0.88169672976012
	Recall	0.8793103448275862
	F1-Score	0.8752647360816025
poly	Test data score	0.7448275862068966
	Accuracy	0.8574712643678161
	Precision	0.8667436874634088
	Recall	0.8574712643678161
	F1-Score	0.8490946311791467

Table 2: SVM Kernels and Evaluation Metrics.

3.2.2 Model Evaluation

Another way that we can evaluate our SVM classification model on unseen data is to use K-Folds cross-validation. This technique involves dividing the available data into multiple folds (subsets), using one fold for validation/test set and the remaining folds for training the model. The process is repeated K times where each time a different fold is used for the validation set. Results from each validation step are then averaged. This technique provides more than one test/train split of the input data, and provides a more robust estimate of the models performance. It divides the data into multiple folds and trains the SVM model on each fold while testing on the others. [Sha23]

The main purpose of cross-validation is to help prevent overfitting of the trained subset data to the model. New unseen data from our validation set could perform poorly on the model producing invalid predictions if the trained subset fit too well to the model. Good results from cross-validation will ensure that the model is robust and generalizes well to new data. [Sha23]

Testing with our SVM model we chose a 30% validation set and 70% training subset; having a

higher percentage for the training subset will ensure that our input bias is minimized as this will include more relevant input features needed to create a more robust model. Table 3 shows K-Fold cross validation metrics where the mean accuracy of the model gets better (closer to 1) at 4 or 5 folds. With the SVM model with kernel=rbf and the one pass train/test split, we generated an accuracy of 0.879 which is better than the 4 or 5 fold results but they are similar to show that the model is robust. Setting K-Folds higher than 5, we didnt notice the mean accuracy change much.

N-Folds	Scores	Evaluation Metrics
2	[0.64 0.82]	0.73 accuracy with a standard deviation of 0.09
3	[0.42 0.92 0.79]	0.71 accuracy with a standard deviation of 0.21
4	[0.64 0.92 0.88 0.79]	0.81 accuracy with a standard deviation of 0.11
5	[0.60 0.87 0.89 0.86 0.80]	0.80 accuracy with a standard deviation of 0.11

Table 3: K-Fold Cross Validation Metrics.

3.3 Best Model Fit

We evaluated each model (kNN and SVM) to see which one predicted the 'completed_task' output best. Figure 6 shows a comparison of prediction results between our SVM and K-Nearest Neighbors on a testing set in the form of confusion matrices. The results of these two models are comparable, where both models performed well at accurately classifying if a task was not completed. One noteworthy difference between the two models is that the SVM was able to identify more true negative results than the kNN model, where it could correctly classify 30 more uncompleted tasks than the kNN model. On the other hand, it mistakenly classified about 50 more tasks as incomplete than the kNN model. So, overall, our SVM could be seen as more pessimistic than the kNN model, since it will tend to classify more tasks as incomplete.

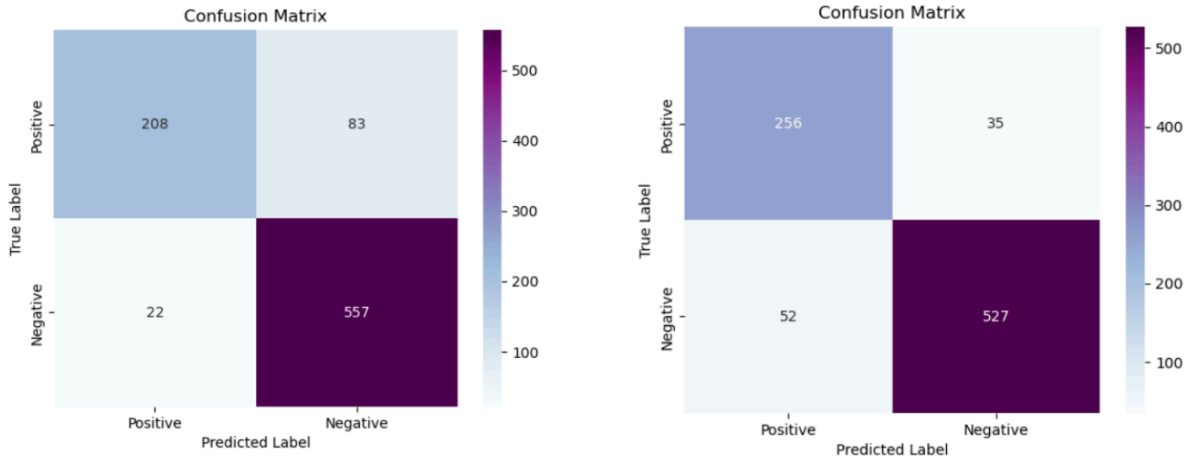

Figure 6: Confusion Matrices: Support Vector Machine (left), K-Nearest Neighbors (right).

3.4 Predictions for Cases of Interest

With cases of interest, we were interested to see how the SVM model would predict 3 new cases of interest based on the following variables: high monetary reward, high assignable HIT counts, and a high labor event count. These are the results we found when examining it.

3.4.1 Case 1: High Monetary Reward

We can see a high amount of monetary rewards (in dollars) offered, along with a low amount of assignable hits and 158 total labor events that occurred within this case. As we can see this task was completed, which our model correctly predicted.

```
# ['task.monetary_reward_in_dollars', 'task.assignable_hits_count', 'total.labor.
    event_count']
case1 = [0.65, 0.0, 158]
case1Result = [1]
```

Listing 5: Case 1 - Feature input and 'completed_task' ouptut.

Case 1 Model Prediction –> **1**

3.4.2 Case 2: High Assignable Hits Count

We can see a high amount of assignable hits count offered, along with $0.08 of monetary reward and 234 total labor events that occurred within this case. As we can see this task was completed, which our model correctly predicted.

```
# ['task.monetary_reward_in_dollars', 'task.assignable_hits_count', 'total.labor.
    event_count']
case1 = [0.08, 5083.0, 234]
case1Result = [1]
```

Listing 6: Case 2 - Feature input and 'completed_task' ouptut.

Case 2 Model Prediction –> **1**

3.4.3 Case 3: High Total Labor Event Count

We can see a high amount of total labor events occur, along with 0.0 amount of assignable hits and 705 total labor events. As we can see this task was not completed, which our model correctly predicted.

```
# ['task.monetary_reward_in_dollars', 'task.assignable_hits_count', 'total.labor.
    event_count']
case1 = [0.08, 0.0, 705]
case1Result = [0]
```

Listing 7: Case 3 - Feature input and 'completed_task' ouptut.

Case 3 Model Prediction –> **0**

4 Summary and Conclusion

TBD

References

[Gig23] GigPlatform. Chrome extension (plugin) that tracks the working time of an amazon mechanical turk work. https://github.com/gigplatform/toloka-web-extension, 7 2023.

[Sha23] Abhishek Sharma. Cross validation in machine learning, 2023.

[TSS21] Carlos Toxtli, Siddharth Suri, and Saiph Savage. Quantifying the invisible labor in crowd work. *arXiv*, 2110.00169(1):1–26, 2021.