ECS 171 Red Team Project Report

Abstract

The ComedAIn machine learning project was made to process a user's demographics and return a set of jokes from the database based on what the program decides would be most enjoyable for the user. The algorithms used in the program are Singular Value Decomposition (SVD) and Random Forest. Vector Space and Neural Networks were also implemented but not used in the final model. When a user first uses the program, SVD finds the suggested jokes based on the user's demographic input. Random Forest is then used to better predict the user's joke rating. Our result is generated from stacking of random forest and SVD model by taking the mean of those two models. We will display first 10 jokes to a new user based on user profile and updated 10 new jokes each time after a user rated the previous 10 jokes. Our method could incorporate jokes inside features, users' profiles, and users' responses all at once, which leads to a better prediction.

1. Introduction

The ECS 171 Red Team is made up of 33 members divided into 5 sub-teams. Throughout the project each sub-team had specific tasks. The Project Manager team successfully established communications through Slack, delegated tasks to the other sub-teams through Trello, and organized the project code on GitHub. The Software Engineering team used Docker to deploy the application on Azure. The project was created using Jinja for the front end and Flask API to connect the web application to the machine learning algorithms and implement user functions. The data used in the application was stored on a SQlite database connected by the Flask API. The Machine Learning team focused on implementhe ting SVD, Random Forest, Vector Space, and Neural Network algorithms, deciding on using SVD and Random Forest in the final project. The Data Visualization team worked with the Software Engineering team to implement the application UI and the Machine Learning team to create plots of the data. Lastly the Quality Assurance team focused on reviewing code quality and helped with the final debugging process. They also created a user experience survey for feedback on the website's design.

The SVD model gives really good prediction on the first 10 jokes because it tends to present the jokes with higher average joke rating which means jokes that everyone will like. There will be a drop of predicted accuracy after the first 10 jokes but the accuracy will increase as user rated more jokes. The drawback of SVD model is that it does not consider the features hide inside the jokes, such as joke type, length of the joke, and so on. Random Forest Model will help to adjust the feature inside jokes and user profile. We will show the results for both methods.

The current approaches we had is that we just simply take mean of both methods, but did not find the best combination of those two. For example, the SVD model gives a really good prediction on the first 10 jokes and relative bad prediction on next 10 jokes based on rating of the jokes. There should be a more weight on SVD model for the first 10 jokes than for the next 10 jokes rather than give it 50

This paper is mainly to present the methods we used in SVD, Random Forest, Vector Space, and Neural Network algorithms and corresponding results and analysis. We try to explain the advantage and disadvantage of each models and figure out why some model works better than the others.

2. Methods

The main goal of our algorithm is to predict the rating of jokes for a incoming new user and provide the user the highest predicted joke ratings based on user's profile and his or her previous joke ratings. After a user rates jokes, we can also predict a user's missing profile, like favorite joke type, favorite music genre, and favorite movie genre based on a user's joke preference.

Multiple methods including SVD, Random Forest,

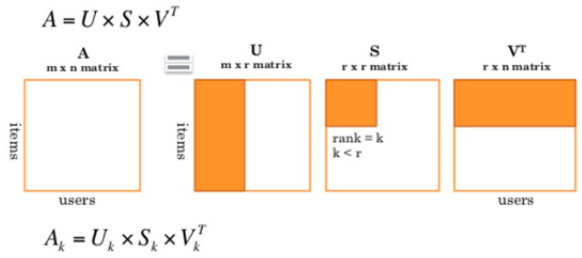
Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

Vector Space, and Neural Network were used, and we will compare the accuracy of the prediction using ROC curve and MSE in next section.

Because of the fact that different models have different strengths, we will apply stacking method to predict our user rating based on multiple models.

2.1. Singular Value decomposition(SVD)

Singular value decomposition is a method of finding the rank-k approximation of a matrix. We utilize this method to find suggested jokes by first creating a matrix which contains all users' features and all of their joke rankings (one column per joke, one row per user). With this matrix, we can input the query vector for a given user that has not rated all jokes into this matrix, then decompose it using SVD to a rank-k approximation. This new rank-k matrix will fill in the missing ratings with what it "believes" or approximates the ratings to be. The new values in this approximated matrix are our estimate for the user's ratings per joke; we use these estimated ratings to find the joke not already rated with the highest rating. In the meantime, if a user failed to input any missing user information like favorite joke type, favorite music genre, and favorite movie genre, SVD will provide an estimate of the user features.



(Figure 1)

In our case, the Matrix A in Figure 1 is our original matrix, each row is corresponding to a joke ID and first 11 columns are user features and from 12 to the last column corresponds to a user ID.

SVD will decompose the Matrix A into 2 singular vector U and V^T and a Singular Diagonal Value S. U and V represent the direction of our features and S represent the magnitude to each directions. We pick the sort the singular value and pick the 10 largest (k=10) singular value to reproduce our matrix A' which is the approximation of matrix A. In our model, estimated A'contains 85% information of original matrix A.

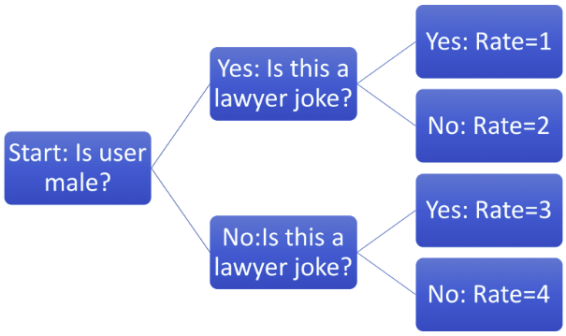
When a new user comes in, we will add a new row in matrix A containing user information only and initial rating of all jokes 0. After doing SVD, we get the estimated A' and from A', we can get the estimated rating of each jokes for that user. After user rated the jokes, the new row in the matrix A will be updated with more information about jokes rating and have more accurate prediction next step.

The advantage of SVD is that it provides approximated rating on all jokes and is robust to noise in the data, as it is dimensional reduction.

There are also some disadvantages. It will be skewed towards more popular jokes; to counteract this we normalize each row to mitigate differences in "user average rating" and normalize each column to mitigate differences in "average rating per joke". This is also completely a collaboration filter model, so it does not take into account any sort of features on the jokes, which we are currently trying to resolve with a vector space model. In the meantime, the computationally complex is O(m^3), however, with a dataset of our size, this should not be an issue. For scalability and larger datasets, an iterative SVD approach could be implemented. We have to choose an optimal k-value for the approximation, which may be hard to find. The approach we are currently looking at is empirically find a k which retains 85% of the original dataset's variance.

2.2. Random Forest

Random forest[6] is an ensemble method that builds multiple decision trees to increase prediction accuracy. Our model random forest will create a set of random decision trees. Each tree will be something like the Figure 2 below,



(Figure 2[5])

We created 50 trees to train the model, with minimizing the square root error methods. After doing feature selections with lasso, our Random Forest model also include additional features from our feature teams, such as '`ave_character`', '`ave_word`', features from $TFIDF$[1,2,3] and so on. Detailed description of additional features as well as exploratory analysis on features from Data Management Team will be in the supplementary file.

The main disadvantage for decision trees is overfitting, but random forests bypass this by using samples of the data to build multiple trees. The regressor uses numerical and categorical (transformed into one-hot vectors) variables to output a continuous predicted score between 1 and 5. Random forests can also output variable importances, and say which variables were most helpful in the prediction. Disadvantages would be that this method does not predict as high as neural networks.

In the end, we picked numerical Random Forest Regressor due to better predictability.

2.3. Vector space (Did not implement it in our final model)

Vector space models are designed to take in a vector comprised of the user features and their rated jokes and run a $K-Means$[8,10] on it to receive back a clustering of these users. Then for the new user, we associate them with one of the clusters and return a list of jokes that have not been seen (voted) yet.

As an output, the algorithm returns a list of vectors for the new user. The vector contains the list of users in the cluster that the new user is in. From there, we can just pull out the jokes that were rated highly by all the users in the cluster and recommend it to the new user.

The advantage of this model is that it provides recommendations based on other users and gives an accurate association between the likeability of a joke and the user. It has the potential for active learning. The disadvantage of this model is that there are not enough users to give an efficient clustering given many joke features. That is the reason why it did not perform really well and we will discuss its results later on.

2.4. Artificial Neural Network(Did not implement it in our final model)

We used the Keras Sequential Model to construct the Artificial Neural Network. The model we chose used Adam as the optimizer and MSE as the loss function. The input layer has 35 input nodes, one for each user attribute (gender, age, etc. as well as indicator/"dummy" variables for each category when applicable). The features we selected were preprocessed as follows:

- Data normalization: We normalized the age attribute to be between 0 and 1 (subtract each value by the minimum age and divide by the range).
- Feature Selection: We decided not to use the features of favorite movie genre or favorite music genre as there no significant benefit to the model's prediction accuracy by using them.
- Dimensionality reduction: We merged certain majors and birth countries (e.g. combining "Computer Engineer", "Computer Engineering", "Electrical Engineering" into one "CE/EE" value), reducing major to 4 possible values and birth country to 3 different values.

Assuming only user features are used as input nodes, there are 2 hidden layers with 10 nodes each. The number of hidden layer nodes must increase as the number of input nodes increases, so as the rater rates more jokes, the number of input layer and hidden layer nodes increase while the number of nods in the output layer decreases. We used ReLU activation function for the first two layers (input to hidden 1, and hidden 1 to hidden 2), and sigmoid as our activation function from the second hidden layer to the output layer.

The output layer has 153 output nodes: one for each joke, with the output of each node being the predicted rating for that joke. Finally, our model was trained using 0.2 as the validation split (meaning 80% for training and 20% for testing). We used 100 epochs and the default batch size of 32.

3. Results

The final model takes the average of predicted rating from SVD and Random forest to get a estimated score when a new user comes in and starts rating. SVD and Random Forest both worked well in terms of predicting the joke rating. We will present the ROC curve for both models. We will also present our finding in other two models, ANN and Vector Space.

3.1. Random Forest and SVD

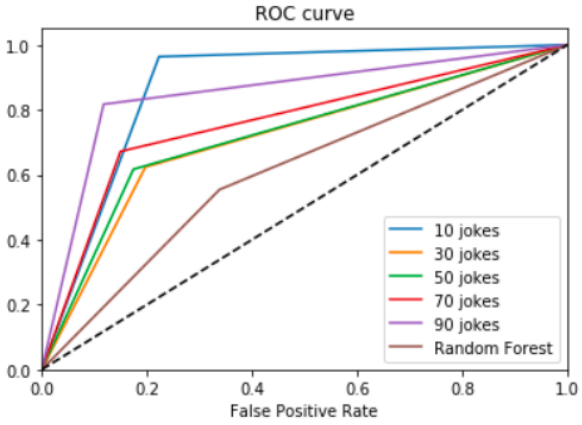
By setting 1 as joke rating bigger than 3 and 0 as joke rating smaller 3, we were able to check the ROC curve given predicted rating and actual rating from both SVD and Random Forest model.

Note: SVD model goes through each user one by one and find all their predicted rating. Random Forest train model with 70% of users and test it with the rest 30% result and we provide the ROC curve of test dataset's result.

Like what we showed in Figure 3, we can see that ROC curve for first 10 jokes using SVD gives a very high AUC, 0.87. The AUC for the rest jokes dropped and goes back as user rated more jokes. It reach a very

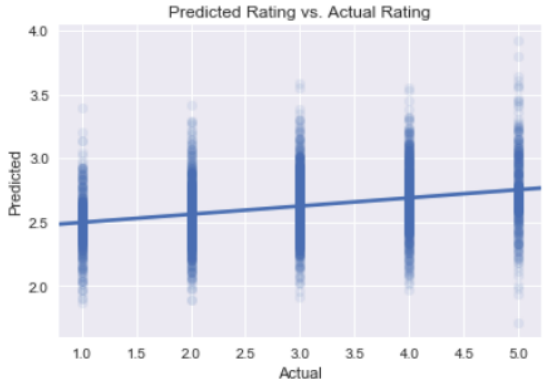
high level. The Random Forest model gives a relative small AUC but still bigger than 0.5.

- Area under the curve for first 10 jokes 0.870321229685
- Area under the curve for first 30 jokes 0.713208820865
- Area under the curve for first 50 jokes 0.721264493532
- Area under the curve for first 70 jokes 0.761003014939
- Area under the curve for first 90 jokes 0.849750052732
- Area under the curve for random forest 0.608031285545



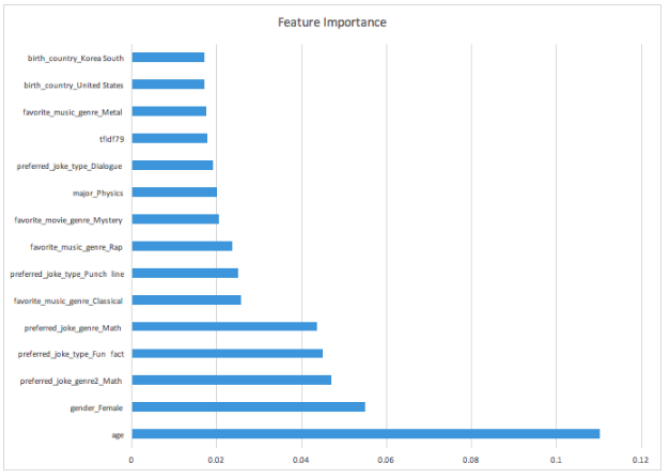
(Figure 3)

For Random Forest, we also checked the numerical rating comparison between actual score and predicted score, shown in the Figure 4 below. In the x-axis it shows actual joke rating from 1 to 5. In the y axis, it shows predicted score from 0 to 5. We also plotted the fitted line for two type of scores.



(Figure 4)

In the Figure 5 below, it shows the top 15 importances of features in terms predicting the joke rating using Random Forest.

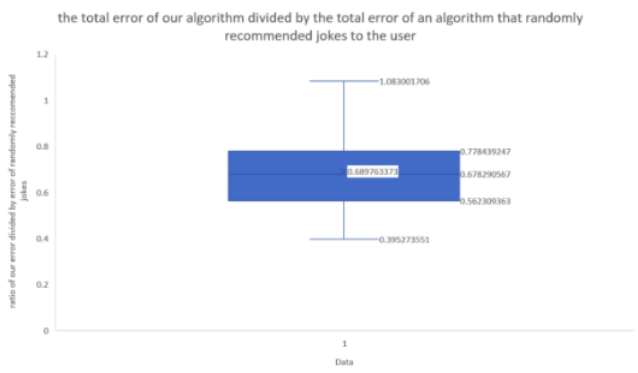


(Figure 5)

3.2. Vector Space model

For Vector Space model, we compared the total error of our algorithm with the total error of an algorithm that randomly recommended jokes to the user. This gives us a ratio to determine how much better our algorithm performs than just randomly recommending users jokes. The closer to 0 the ratio is, the better our algorithm performs, since there are less errors in our algorithm than the errors of just randomly guessing. Getting a ratio of 1 means that our algorithm is as bad as randomly recommending jokes. Most of our tests resulted in greater than a .5 ratio and the average was 0.689763373.

The Figure 6 below shows the medians (Q1,Q2,Q3) of the total error of our algorithm divided by the total error of an algorithm that randomly recommended jokes to the user. It also gives the mean of that. It seems that 75% of the ratios lie above .56.

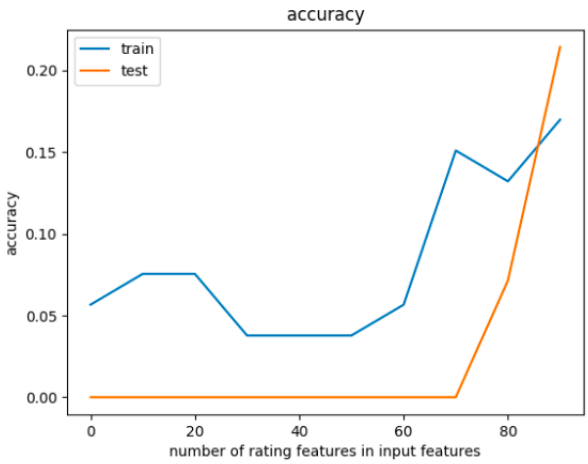


(Figure 6)

3.3. ANN

For ANN, our model was trained using 0.2 as the validation split, meaning 80% for training and 20% for testing.

The Figure 7 below from both the Rate of Change plot of our ANN model. As labeled, the blue line represents the accuracy as the number of input rating features increase for the training set, and the orange line represents the same for the testing set. Note: the features number increased as the user rated more jokes.



(Figure 7)

4. Discussion

Our final model is combining SVD and Random Forest to predict joke ratings. We will discuss why ANN and Vector Space model failed and the credibility of our final models.

4.1. SVD and Random Forest

The SVD provide a really good prediction on the first 10 jokes and the accuracy when down after that but goes back each times rater rated more jokes. Random Forest gives a comparable small AUC of 0.6 which is still better than random guess. The reason why it predicts perform less accurate is that it does not take the mean rating of each joke into account. That is why it had some hard time finding the "popular jokes". However, the model is purely focused on the features of user and jokes, which gives our model a different aspect of the picture. We believe that after combining with SVD model, it could incorporate both aspect of popularity and jokes's feature in our final model. The proportion of those two models need to be consider in the next step, but we did not included in this paper.

Overall, both SVD and Random Forest gives a very good predictions and we can present user the jokes they will like.

4.2. ANN and Vector Space

We decide to reject Vector Space Model and ANN model.

For Vector space model: after finding the ratio of the total error of our algorithm divided by the total error of an algorithm that randomly recommended jokes to the user, the Figure above shows the medians (Q1,Q2,Q3) of the ratios and 75% of the ratios lie above .56, which is not a good indication for our prediction accuracies. Most of our tests resulted in greater than a .5 ratio and the average was 0.689763373, which meant that our algorithm is close to being as bad as randomly recommending jokes. We decide to reject this Vector Space model.

The reason why it failed may because we have not enough users to give an efficient clustering given many joke features. We could improve our model when there are more raters that rated our jokes.

For ANN model, the accuracy plots in Figure shows that for both training and test, there was a really low accuracy. As more features came in, the accuracy went up a little but still at a very low level of about 10%.

The reason why it failed was because of dimensionality and retraining. Similar like Vector space model, ANNs require a lot of training data, especially for high-dimensional feature space. As the user rates more jokes, the feature space dimension increases for that user. The number of features may even exceed the number of data samples. Also after a user rates a joke, the user's rating for that joke becomes a new input node for the ANN of that user, and the joke is removed from the output node. This results in a new model with a different number of input and output nodes. Thus the ANN would have to be retrained each time.

5. References

1. Using TF-IDF to Determine Word Relevance in Document Queries, Juan Ramos, 2003 (Used in supplementary file)

2. NLTK documentation, http://www.nltk.org (Used in supplementary file)

3. Stanford Natural Language Processing group, https://nlp.stanford.edu/software/ (Used in supplementary file)

4. DATA 643 Project 3, Logan Thom-

son, June 24 2017 https://rstudio-pubs-static.s3.amazonaws.com/287384_361b4e0daf5648fcafb404cade2e2bc1.htm

5. Random Forests in Python, June 5, 2013

6. Introduction to Random forest, Raghav Aggiwal, Feb 28, 2017

7. Pytrends, https://github.com/GeneralMills/pytrends (Used in supplementary file)

8. Exploiting User Demographic Attributes for Solving Cold-Start Problem in Recommender System , Laila Safoury and Akram Salah , August 2013

9. Recommendation Systems, Jeffrey D. Ullman http://infolab.stanford.edu/~ullman/mmds/ch9.pdf

10. Comparing Python Clustering Algorithms, Leland McInnes, John Healy, Steve Astels, 2016 http://hdbscan.readthedocs.io/en/latest/comparing_clustering_algorithms.html

6. Author Contributions

6.1. Machine Learning Team

6.1.1. Algorithms

1. Kurt Schneider (kmsch@ucdavis.edu):
Leader of algorithm group. In charge of communicating between groups, building SVD model, stacking SVD and Random Forest, holding Algorithm meeting, keeping records of meetings. SVD is included in our final model and provided a detailed description of the model.

2. Ari Schoenfeld (abschoenfeld@ucdavis.edu):
In charge of building SVD model. Included in our final model and provided a detailed description of the model.

3. Megha Jain (mrjain@ucdavis.edu):
In charge of building ALS model, but couldn't get results out because of some bugs and ALS model is not included in this paper.

4. Cynthia Lai (cynlai@ucdavis.edu):
In charge of building Random Forest. She also provided a lot of great materials and paper to read and start collecting and braining storming ideas of algorithms. She also incorporated active learning, feature team's features, and data cleaning (nans, low variance) in the Random Forest.

5. Allen Speers (atspeers@ucdavis.edu):
In charge of building vector space model. Vector space is not included in our final model but he provided a detailed description and the reason why it fails.

6. Richard Hou (rhhou@ucdavis.edu):
In charge of building Random Forest model. Random Forest is included in our final model and he provided a detailed description of the model.

7. Jason Liu (jymliu@ucdavis.edu):
In charge of building ANN. ANN is not included in our final model but he provided a detailed description and the reason why it fails.

8. Melanie Zhang (mlnzhang@ucdavis.edu):
In charge of building ANN. ANN is not included in our final model but he provided a detailed description and the reason why it fails.

9. Calvin Huang (calhuang@ucdavis.edu):
In charge of building vector space model. Vector space is not included in our final model but he provided a detailed description and the reason why it fails.

6.1.2. Features

1. Sailesh Patnala (sgpatnala@ucdavis.edu):
Leader of feature group, in charge of NLP, holding Feature meeting, communicating with software engineer team to implements Machine Learning algorithm with our backends. That was a lot of works and he did a great job on it.

2. Joe Akanesuvan (kakanesuvan@ucdavis.edu):
In charge of finding trending locations of the keywords in joke text, used the list of keywords extracted from Sailesh's NLP and used Google Trends API to get trending location from keywords.

3. Chia-Hui Shen (chshen@ucdavis.edu):
In charge of doing TF-IDF. Also she did NLP and found number of words in sentences and average number of character in words. She is very active in our team and get things done very fast.

4. Jonas Lum (jmlum@ucdavis.edu):
In charge of combining features and found potential ideas of data cleaning.

5. Sadegh Shamsabardeh (mshamsabardeh@ucdavis.edu):
In charged of finding std Deviation of the joke, determine decisiveness of the joke and normalized rating based on users.

6.2. Data Visualization

1. Felix Le (fple@ucdavis.edu):
   In charge of making the data visualizations

2. Injung Ahn (injahn@ucdavis.edu):
   In charge of making the data visualizations

3. Anthony Ho (aawho@ucdavis.edu):
   In charge of making the data visualizations

6.3. Software Engineering

1. Andres Sanchez (anisanchez@ucdavis.edu):
   Updating the routing mechanism for the website and created multiple pages on the side. In charge of many features in the beginning of the process.

2. Maurice Dela Fuente (mdelafuente@ucdavis.edu):
   Worked heavily with other engineers during the setup process to insure a clean architecture. Found and solved bugs in the system.

3. Matthew Kyawmyint (mmyint@ucdavis.edu):
   Led the home page implementation effort, and solved critical bugs on the application. Refactored critical SVD code into a class for easier understanding.

4. Eden Bernabe (eabernabe@ucdavis.edu):
   Worked together alongside other engineers on the feature development process. Found and solved bugs in the system.

5. Jason Hui (jashui@ucdavis.edu):
   Moved the database classes into the codebase and created models. Perfected our database experience and heavily worked alongside other engineers.

6. Bryan Kim (bjkim@ucdavis.edu):
   Worked on building multiple features alongside other engineers. Built a 'Guess Me' feature and kept our packages updated.

7. Alex Derebenskiy (avderebenskiy@ucdavis.edu):
   Implemented multiple features on the site, including showing the previous joke ratings of a user. Worked on other fixes such as moving our database to a specific encoding.

8. Jisoo Yun (jisyun@ucdavis.edu):
   Led all deployment efforts of the project. Dockerized the application and put it onto an Azure server. Refactored code to work with Docker.

9. Daniel Velasquez (dpvelasquez@ucdavis.edu):
   Built some of the most difficult features on the site, including the entire joke page. This included automatic joke fetching and showing likelihood weighting. Designed certain aspects of the model architecture. Solved critical bugs in the ML implementation.

6.4. Quality Assurance

1. Matthew Corbelli (mdcorbelli@ucdavis.edu):
   In charge of testing code and searching for bugs in the application

2. Akshay Kumar (thekumar@ucdavis.edu):
   In charge of ensuring a quality standard throughout all of the project's code

6.5. Project Management

1. Kai Jin (kchjin@ucdavi.edu):
   In charge of managing machine learning team. Also contributed to the project reports and presentations.

2. Carson Dacus (cbdacus@ucdavis.edu):
   In charge of managing the quality assurance team. Also contributed to the project reports and presentations.

3. Stephan Zharkov (sdzharkov@ucdavis.edu):
   In charge of managing the software engineering team. Set up the initial architecture of the project such as the initial design and database setup. Controlled all pull requests made on Github to avoid conflicts, and heavily tested code. Worked on multiple features and the final utilization of ML algorithms. Also contributed to the project reports and presentations.

4. Melody Chang (mccchang@ucdavis.edu):
   In charge of managing the data visualization team. Also contributed to the project reports and presentations, wrote the entire final report in LaTeX.