First Impression for Dummies

### **Problem Statement**

Swipe left… Swipe left… Swipe right… MATCH! With applications such as Tinder, Linkedin, and Facebook, users can form impressions on several people in a short amount of time which has greatly affected the way we date, network, and make friends in the 21st century. According to Anna Pitts of Business Insider, first impressions are made in just 7 seconds. This is precisely the problem we aim to address: What are the characteristics that help someone nail a first impression? Using a dataset gathered from a speed dating event, we were able to run various models to see if we could predict if a participant received a match based on the participant’s personality and how the participant came across as. The results from our study provides deeper insights on how we can better market ourselves which has the effect of improving our relationships, job prospects, and friendships.

### **Solution**

### **Data used & Clean Up**

As mentioned before, the dataset is gathered from a speed dating event. Participants were asked several questions about themselves, their date, and an indication if the participant received a match or not. Originally, there were 123 features. About half of these features had duplicate information as the dataset has a column for the original numbers and another column that has those values in categorical bins. We were more interested in the numeric value for features so we decided to remove all other variables that were binned. After using pandas to delete the duplicate columns, we were left with 67 features. A portion of these features dealt with information about the participant’s personal details which includes age, race, and job, the participants interest in different activities such as hiking, shopping, etc., and what the participant valued in a partner such as sincerity or intelligence, etc.. Another portion of the dataset contained information about what the participant thought of their partner, and what the partner thought of the participant. The final column was whether the participant matched with their partner at the end of the speed dating event. For numerical variables, we replaced the null values with the mean of the variable. For categorical variables, we replaced the null values with the mode of the variable. Originally, there were 8378 records. However, some records had too many null values and we decided to delete the record completely. This brought us to a total of 8220 records.

##### **Feature Engineering:**

##### **One Hot Encoding**

Before we started on feature engineering, one thing we did with the categorical variables in our dataset is to one hot encode them into numerical variables. We chose one hot encoding instead of label encoding as our variables does not have an ordinal relationship, and we did not want the classifiers to think that one variable is better than another because the number given is higher. However, this comes with a price as the number of features increased by the same number of categorical variables we had. To remedy the large number of features we gained, we used principal component analysis (PCA) to reduce the number of features we have.

### **Undersampling/Oversampling**

Even though we had 8220 records, the majority class with 6870 records did not have a match, whereas the minority class with 1350 records did have a match. This causes a problem as the classifiers would overlook the minority class and give misleading accuracies that favors the majority class. To balance our dataset, we tried two different methods: undersampling and oversampling. We decided to use oversampling as we did not want to lose the data we had from the majority class. The oversampling technique we employed is synthetic minority over-sampling technique (SMOTE), and we tuned it to resample only the minority class to create a balanced dataset with 13740 records which is used in our analysis.

### **Feature Creation**

Looking at the numbers provided by the participants such as importance and ratings, we decided to create new features that compares the numbers provided by the participant and their partner. Some of the features created are the difference between the participant and their partner, the difference between the rating the partner gave the participant and the rating the participant gave themselves, and the difference between the importance and rating for both the participant and their partner. As the scale for rating over the different categories and activities is independent and from 1-10, and the scale for importance over the different categories add up to a total of 100, we created new features that changed the scale for rating and activities to add up to a total of 100 as well to make the comparison between importance and rating possible.

#### **Approaches & Assumptions**

For each form of analysis, we used an 80/20 training and testing split. Some of the models used a PCA version of our data which brought our 375 dimensional space down to a 253 dimensional space.

Decision Tree

Using the PCA version of our data and a 5 fold cross validation, we saw 96% training accuracy, 78% test accuracy, and 79% for precision, recall, and f1.

Random Forest

Using a 5 fold cross validation and the PCA version of our data, we saw 99% training accuracy, 91% test accuracy, and 91% for precision, recall, and f1.

AdaBoost

AdaBoost short for adaptive boosting is an ensemble classifier on sklearn that is a meta-estimator that starts by fitting a classifier on the dataset and then fits addition copies of the classifier with different weights on the same dataset. We included this in our analysis as we wanted to be able to have a comparison between the different boosting ensembles.

Gradient Boosting

Gradient Boosting (GBM) differs from Random Forest by the order the trees are built and the way results are combined. GBM uses regression trees and it works well with imbalanced datasets by strengthening the impact of the minority class. GBM are better learners than Random Forest and would perform better if the hyperparameters are tuned. However, it is more prone to overfitting compared to Random Forest.

Extreme Gradient Boosting

Extreme Gradient Boosting (XGBoost) is an advanced implementation of GBM. XGBoost uses a regularized model formalization to control overfitting, implements parallel processing, and is able to deal with irregular datasets. Darwin uses XGBoost as their GradientBoosting algorithm and we decided we should explore this classifier as well.

K Nearest Neighbors

K nearest neighbors (KNN) is a classifier that looks at the k number of neighbors around a record and the record gets classified based on the votes of said neighbors. This classifier is susceptible to the curse of dimensionality and therefore we had to scale the data and perform PCA before using this. As our dataset has a high dimensionality, this classifier is not a good choice for our analysis.

Support Vector Machine

Because there were around 250 attributes present in the dataset, this algorithm particularly struggled to find the best kernel. SVM is one of the most widely used algorithm and is also not susceptible to the curse of dimensionality. Despite being such an appealing algorithm, SVM only produced an accuracy around 67.5% on SMOTE data. The f-1 score of correctly predicting a matched person was in low 20%.

Naïve Bayes

With a 10-fold cross-validation, Naïve Bayes received an accuracy around 48% with f-1 scores also around 50% mark. When randomly guessing an outcome between yes/no, one has the probability of getting 50% of the answers correct. Therefore, we are no better than guessing the outcome with the Naïve Bayes model since it gets only half of the predictions correctly.

Neural Networks

Neural networks is a multi-layer perceptron classifier that has at least one hidden layer to learn complex and diverse decision boundaries. It is not susceptible to the curse of dimensionality as irrelevant features could be allocated to no weight in the classifier. However, it is susceptible to overfitting, the training time is long and requires a lot of data. This is one of the three models that Darwin uses. In our analysis, we changed hyperparameter of the hidden units in a single hidden layer but Darwin was able to calculate an optimal hidden layer number and the appropriate number of hidden units in each layer in a much shorter time.

#### **Application of Darwin**

We used Darwin as it was able to clean the dataset and create a model to help us predict whether a participant will receive a match, and Darwin does this in no time at all. We tried using Darwin to clean our original dataset but we were running into an error that says that Darwin could not process it. We then went back to clean up some of the data on our own, balanced the dataset using SMOTE and split it into training and testing datasets before uploading it to Darwin. Looking at the datasets we uploaded to Darwin after we cleaned it and datasets after Darwin cleaned it, there was no further cleaning from Darwin’s part as the dataset are the same. We also used Darwin to create a model to predict the result we wanted to get. Darwin uses 3 different model types: DeepNeuralNetwork, RandomForest, and GradientBoosting. If we did not specify a model type, Darwin would use the model with the best model accuracy and display the results. We explored all 3 model types with our training and testing datasets and realise that even though Darwin selects DeepNeuralNetwork as the best model, the accuracy it shows on the testing dataset does not do as well as the RandomForest or GradientBoosting.

We enjoyed using Darwin as it significantly reduced the time needed to get the result from the classifiers. Darwin’s display\_population also provides the hyperparameters selected for each model and we were able to compare it with our own findings as well. However, using the hyperparameters from Darwin, we ran an analysis on our training and testing datasets on Jupyter and notice that there is a discrepancy in the accuracy for the testing dataset. The testing accuracies and f-1 scores differ significantly from our own models which raises the concern whether Darwin’s models are overfitting on the training data. Darwin should work on updating the dataset they have when the user uploads a dataset with the same name because it currently throws an error and it would be better if Darwin updates the dataset they have internally rather than having the user delete and reupload their dataset. Darwin could also expand on working with imbalanced datasets as most datasets in the real world are imbalanced. Darwin should also be able to split the given dataset into training and testing datasets internally, create the model based on only the training dataset and do predictions on both.

### **Team Engagement**

We were able to make time, about once a week, to sit down and discuss about the project and about the direction we were going to take. In order to reduce the amount of time running every classifier multiple times with hyperparameter tuning, we decided that each member will take on some of the classifiers we named above and do our own individual analysis on it. We met up to discuss what we analyzed and combined our analysis into one final analysis. We also explored the different aspects of Darwin and what Darwin was doing individually as we were unfamiliar with the Darwin SDK. We also decided to divide the report so that each member is able to further explain their allocated classifiers. Each member was able to complete what was allocated to them and are able to explain what they did in their analysis that helped to create our final analysis.

### **General Challenges**

The first challenge we dealt with was finding a balance between time and hyperparameter tuning. For each model, we try out various parameters that would yield the best results. Unfortunately, adding too many parameters took too long for the code to finish executing. We used trial and error to have enough parameters while still executing our code in a reasonable amount of time. The next challenge we ran into was how Darwin handled our datasets. We wanted to feed Darwin two versions of our data: cleaned dataset with SMOTE and cleaned dataset without SMOTE. Darwin was able to develop models and metrics for our SMOTE training and testing dataset, it was also working for our training data without SMOTE, but it was running into ‘ModelRunError’ or ‘DarwinInternalError: uncaught’ when trying to run model with the testing dataset without SMOTE. We showed this issue with Sari from Darwin but he was unable to fix it, hence we decided to leave out the no SMOTE Darwin analysis.

### **Next Steps**

Looking at feature\_importance, we saw that the attributes that played a major role in a participant receiving a match were attractiveness, funniness, intelligence, sincerity and ambitiousness. Our next step involves how we can capitalize on these features on a dating profile. For example, if we were to guide a user to improve their profile, we would first highlight funniness with a quirky joke in the biography or a light-hearted picture. We would then highlight intelligence by including the user’s educational intuition and job title, showcase sincerity by including a picture of the user volunteering or spending time with puppies, and then finally exhibiting ambitiousness by including a picture of the user exploring a landmark or hiking a mountain. Making sure that an important feature is appropriately projected before moving on to the next important feature puts a user using a dating application in the best position to not only stand out in a large pool, but also nail the first impression and meet someone truly special. We can extend the results from feature\_importance and guide those looking for jobs and/or friends to first focus on projecting their intelligence, sincerity, and then ambitiousness. Future research will involve finding datasets with various features of job applicants who were either accepted or rejected for a job. We plan to use the tools in Darwin and the lessons we learned from this project to further explore how someone can better market themselves in a job recruiting setting.