1. INTRODUCTION

1.1 Context and Aims

Online dating is becoming an increasingly popular method to meet a potential partner, with 1 in 5 committed relationships now beginning online[1]. Sites such as eHarmony, Tinder and Match.com use a range of techniques to help users find a potential match. Most systems compare location and age and often users with similar preferences will be paired. The aim of this project was to build a predictive model that could help improve the pairing mechanic of these websites.

By improving match success, these companies can deliver a more tailored service and boast a better success rate for users, increasing their value and popularity.

1.2 Choosing a Dataset

The selected dataset had to contain features similar to those seen on dating platforms. The *Speed Dating Experiment*[2] is a dataset formed from a series of experimental speed dating events between 2002-2004. Participants were asked to evaluate their own hobbies, interests and demographics along with their desired characteristics in a partner, as well as report the success of a number of dates they took part in. This success was expressed as a match or no-match, where the output of both people believing they were a successful match leading to a '1' and all other outcomes being a '0'.

1.3 Data Preparation

The provided dataset contained over 8000 instance rows of numerical data, strings and NaN cells, and 190 attribute columns. The data had to be formatted before using it for predictive analysis. Since each instance was repeated for both partners, the actual number of instances was halved with the useful information from the second instance being placed in new columns within the first instance using Excel's Lookup function. All non numerical data was categorised and converted into integers and all incomplete instances were deleted to make the csv file readable with Python. As a comparison was to be made between partners preferences, new features were created that compared their individual data such as: binary comparisons for profession, religion and race, and difference comparisons between hobbies, interests and partner attributes.

As can be seen in Figure 1 the dataset was unbalanced between matches and no-matches; this was first noticed when the confusion matrix was only predicting negatives (no-match) in an early test model. Undersampling was conducted in order to balance the dataset to a uniform 50-50 match to no-match, eliminating the 100% no-match scenario. The remaining data was split 60-20-20 to create train, validation and test datasets; the validation dataset was used to help reduce overfitting and the training data was used to standardise these sets using the mean and standard deviation.

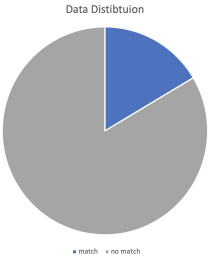

Figure 1: Initial Match Distribution

	coefficient	std	p-value	[0.025	0.975]
intercept	-1.671	0.163	0.000	-1.992	-1.349
int_corr	0.368	0.432	0.395	-0.482	1.218

Confusion Matrix (total:412) Accuracy: 0.862
TP: 0 | FN: 57
FP: 0 | TN: 355

Figure 2: Confusion matrix from an early model

2. PREDICTIVE MODEL

2.1 Model, Results and Discussion

With a large number of potential variables influencing a successful match, it was difficult to determine which variables to use in a predictive model. Therefore, a forward selection algorithm was used to construct a list of features that had the strongest correlations to a successful match. In order to restrict the probability of a match being between 0 and 1, a logistic regression model was used instead of a linear model. It was determined that factors such as similar race (p=0.206), similar profession (p=0.974) and age difference (p=0.062) had weak but stronger correlations with successful matches than most common hobbies. At this stage the model had an accuracy of 55.6% on training data, 64.7% on validation and 59.5% on the test data. Whilst this appears to be quite low, it confirms that the model is better than randomly pairing partners together which would have an accuracy of 50%. After implementing a logistic lasso regression model, some of the variables were identified to have negligible impact on the performance of the model. To set this up this, 'penalty = 'l2' and C = 0.2' was used to lasso the weakest variables and set

	coefficient	std	p-value	[0.025	0.975]
intercept	0.000	0.124	1.000	-0.244	0.244
imprace_diff	0.000	0.120	1.000	-0.237	0.237
age_diff	-0.220	0.131	0.092	-0.477	0.037
go_out_diff	-0.111	0.108	0.301	-0.323	0.101
sin_diff	0.041	0.152	0.787	-0.257	0.339
tvsports_diff	0.000	0.126	1.000	-0.247	0.247
sports_diff	-0.006	0.135	0.965	-0.271	0.259
career_diff	0.000	0.131	1.000	-0.258	0.258

Confusion Matrix (total:272) Accuracy: 0.614
TP: 90 | FN: 42
FP: 63 | TN: 77

Figure 3: Results of Final Model

their model coefficients to 0. This meant that the remaining variables included age difference, opinions of sports and going out, as well as the importance of sincerity in a partner.

Whilst this process slightly reduced the accuracy of our model on the validation and test data to 63.1% and 59.2% respectively, it was a positive step in simplifying the model, reducing the chance of overfitting. As can be seen from the confusion matrix, the total number of true positives was 90/272 (≈1/3), therefore implying that a dating software using this model should guarantee a potential match for every three partner profiles it shows the user.

Alternative models were also tested on the data including random forests and decision trees however no improvement was seen in the accuracy of the model.

2.2 Discussion and Conclusion

The accuracy score achieved with this data could reflect the idea that when someone chooses a match, it is an instinctive decision; it could be predicted that two people are extremely likely to match and common interests are a factor, however the ultimate decision is likely to be a human gut feeling, which is inherently random and difficult to predict. To improve our results, a larger dataset that involves more of the population as well as using more data about each user, such as details about their past relationships and lifestyle choices, could be used.

Having carefully considered and analysed our test data, it was determined that age difference, enjoyment of going out and sincerity were the most important factors to consider when predicting a successful match. Our final accuracy, precision and recall on the test data was 0.59, 0.59 and 0.51 respectively. When applied to a real world scenario, the P-value for the 'match' threshold will be adjusted to

minimise False Negatives in favor of False Positives. This reasoning comes from the idea that the user would rather experience a bad date than not be given the chance to experience a good one.

END OF REPORT

3. APPENDIX

3.1 References

[1] DatingNews.com. (2019). *Online Dating Statistics & What They Mean for the Future of Dating*. [online] Available at: https://www.datingnews.com/industry-trends/online-dating-statistics-what-they-mean-for-future/ [Accessed 23 Jun. 2019].

[2] Montoya, A. (2019). *Speed Dating Experiment*. [online] Kaggle.com. Available at: https://www.kaggle.com/annavictoria/speed-dating-experiment [Accessed 23 Jun. 2019].

3.2 The Code

Importing Libraries

```
import copy
import warnings
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix
```

Warnings and loading the file

```
warnings.filterwarnings("ignore")
dates = pd.read_csv("speed_dating.csv")
dates.head()
```

Balancing the dataset

```
total = len(dates)
nb_pos = dates['match'].sum()
nb_neg = total - nb_pos
data_pos = dates.loc[dates['match'] == 1]
data_neg = dates.loc[dates['match'] == 0].sample(nb_pos)

resampled_data = pd.concat((data_pos, data_neg))
#resampled_data.head()

len(resampled_data)
```

Replacing NaN values with 0s and splitting data into Train, Validate and Test

```
df = pd.DataFrame(data=resampled_data)
df.fillna(0, inplace=True)
print(len(df))
train, other = train_test_split(resampled_dating,
                                test_size=0.4, random_state=0)

validation, test = train_test_split(other,
                                    test_size=0.5, random_state=0)
```

```
print('The sizes for train, validation and test are {}'
      .format((len(train), len(validation),len(test))))
```

```
The sizes for train, validation and test are (816, 272, 272)
```

Creating a list of attributes to omit from the model, mostly columns referring to individual participants instead of comparisons between partners

```
toDrop = ["iid", "id","pid","gender","idg","condtn","wave", "round","order",
          "partner","match","age","age_o","race_o","pf_o_att","attr1_1","pf_o_sin","sinc1_1","pf_o_int",
          "intel1_1","pf_o_fun","fun1_1","pf_o_amb","amb1_1","pf_o_sha","shar1_1","field_cd","field_cd_o",
          "imprelig","imprelig_o","goal","goal_o","date","date_o","go_out","go_out_o","sports","sports_o",
          "tvsports","tvsports_o","exercise","exercise_o","dining","dining_o","museums","museums_o","art",
           "art_o","hiking","hiking_o","gaming","gaming_o","clubbing","clubbing_o","reading","reading_o",
           "tv","tv_o","theater","theater_o","movies","movies_o","concerts","concerts_o","music","music_o",
           "shopping","shopping_o","yoga","yoga_o","exhappy_o","exphappy","int_corr","coef"]
```

Selecting 'match' column to predict

```
X_train = train.drop(columns=toDrop)
y_train = train['match']

X_val = validation.drop(columns=toDrop)
y_val = validation['match']

X_test = test.drop(columns=toDrop)
y_test = test['match']
```

Standardising the data against the training data

```
# Standardise the splits.

X_means = X_train.mean(axis=0)
X_stds = X_train.std(axis=0)

X_train = (X_train - X_means) / X_stds
X_val = (X_val - X_means) / X_stds
X_test = (X_test - X_means) / X_stds
```

Automatic forward selection for most significant variables

```
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

def my_fwd_selector(X_train, y_train, X_val, y_val):
    print('================ Begining forward selection =================')
    cols = list(X_train.columns)
    best_val_acc = 0
    selected_vars = ["imprace_diff"]
    while len(cols) > 0:
        print('Trying {} var models'.format(len(selected_vars) + 1))
        candidate = None
        for i in range(len(cols)):
            current_vars = selected_vars.copy()
            current_vars.append(cols[i])
            if len(current_vars) == 1:
                new_X_train = X_train[current_vars].values.reshape(-1, 1)
                new_X_val = X_val[current_vars].values.reshape(-1, 1)
            else:
                new_X_train = X_train[current_vars]
                new_X_val = X_val[current_vars]

            mod = LogisticRegression(C=1e9).fit(new_X_train, y_train)
            val_acc = accuracy_score(y_val, mod.predict(new_X_val))
            if val_acc - best_val_acc > 0.005:
                candidate = cols[i]
                best_val_acc = val_acc
        if candidate is not None:
            selected_vars.append(candidate)
            cols.remove(candidate)
            print('------- Adding {} to the model ---------'.format(candidate))
        else:
            break
        print('Columns in current model: {}'.format(', '.join(selected_vars)))
        print('Best validation accuracy is {}'.format(np.round(best_val_acc, 3)))
    return selected_vars
```

Applying forward selection function on training data, testing on validation

```
selected_columns = my_fwd_selector(X_train, y_train, X_val, y_val)
```

Create Lasso Logistic Regression and print results for each data set

```
# Lasso Logistic Regression
model = LogisticRegression(penalty='l1', C=0.2).fit(X_train[selected_columns], y_train)


y_train_predicted = model.predict(X_train[selected_columns])
y_val_predicted = model.predict(X_val[selected_columns])
y_test_predicted = model.predict(X_test[selected_columns])

print()
print('======= Accuracy  table =======')
print('Training accuracy is:    {}'.format(accuracy_score(y_train, y_train_predicted)))
print('Validation accuracy is:  {}'.format(accuracy_score(y_val, y_val_predicted)))
print('Test accuracy is:  {}'.format(accuracy_score(y_test, y_test_predicted)))
print()

print('======= Precision table =======')
print('Training precision is:    {}'.format(precision_score(y_train, y_train_predicted)))
print('Validation precision is:  {}'.format(precision_score(y_val, y_val_predicted)))
print('Test precision is:  {}'.format(precision_score(y_test, y_test_predicted)))
print()


print('======= Recall table =======')
print('Training recall is:    {}'.format(recall_score(y_train, y_train_predicted)))
print('Validation recall is:  {}'.format(recall_score(y_val, y_val_predicted)))
print('Test recall is:  {}'.format(recall_score(y_test, y_val_predicted)))

cm = confusion_matrix(y_test, y_test_predicted, labels=[1, 0])
print()
print(cm)
```

Analysing the model

```
model = LogisticRegression().fit(X_train[columns_in_model], y_train)
modsummary = ModelSummary(model, X_val[columns_in_model], y_val)
modsummary.get_summary()

print()

testmodsummary = ModelSummary(model, x_test[columns_in_model], y_test)
testmodsummary.get_summary()

print()

modsummary = ModelSummary(model, X_train[columns_in_model], y_train)
modsummary.get_summary()
```

Adjusting the P threshold to favour False Positives

```
pred_probas = best_model.predict_proba(X_val)

print('Dimension of the predicted probabilities array: ', pred_probas.shape)

# selects probabilitys of class being equal to 1
pred_probas = pred_probas[:, 1]

threshold = 0.7

y_pred = pred_probas > threshold

validation_accuracy = accuracy_score(y_val, y_pred)
print('Validation score using a treshold of {}: {:.2%}'.format(threshold, validation_accuracy))
```

3.3 Alternative Models

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

best_model = None
max_validation_accuracy = 0

for n_estimators in [2, 4, 6, 8, 10]:
    model = RandomForestClassifier(random_state=0, n_estimators=n_estimators,
                                   max_depth=4, min_impurity_decrease=0.001)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    accuracy = accuracy_score(y_val, y_pred)
    print('Number of trees: {}, accuracy: {} '.format(n_estimators, accuracy))
    if accuracy > max_validation_accuracy:
        max_validation_accuracy = accuracy
        best_model = model

print('Optimal number of trees: {}'.format(len(best_model.estimators_)))
```

```
Number of trees: 2, accuracy: 0.5330882352941176
Number of trees: 4, accuracy: 0.5404411764705882
Number of trees: 6, accuracy: 0.5735294117647058
Number of trees: 8, accuracy: 0.5514705882352942
Number of trees: 10, accuracy: 0.5477941176470589
Optimal number of trees: 6
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score

# your code here
dt = DecisionTreeClassifier(max_depth = 2) # Our classification tree
dt = dt.fit(X_train, y_train)

print('\nFor the validation set:')
print('Accuracy: \t{}'.format(accuracy_score(y_val, dt.predict(X_val))))
print('Precision: \t{}'.format(precision_score(y_val, dt.predict(X_val))))
print('Recall: \t{}'.format(recall_score(y_val, dt.predict(X_val))))
```

```
For the validation set:
Accuracy:       0.5073529411764706
Precision:      0.4957627118644068
Recall:         0.8863636363636364
```