Project 1:

Predicting Match in Speed Dating

Content

- Introduction
- Data Preprocessing
  - Feature Selection
  - Multicollinearity
  - Imbalanced Data
- Modeling
  - Logistic Regression
  - Decision Tree
  - XGBoost

Introduction: Predicting Romantic Matches

- Dataset source: https://www.kaggle.com/datasets/ulrikthygepedersen/speed-dating?select=speeddating.csv
- This project aims to predict if two people will match using a dataset gathered from experimental speed dating events from 2002-2004.
- The dataset includes information from participants who had a four-minute "first date" with every other participant of the opposite sex, and were asked to rate their date on attributes including: Attractiveness, Sincerity, Intelligence, Fun, Ambition, and Shared Interests.
- Participants were also asked whether they would like to see their date again and filled out questionnaires on demographics, dating habits, self-perception, beliefs on what others find valuable in a mate, and lifestyle information.

Data Preprocessing: Feature Selection

```
df.shape
```

```
(8378, 123)
```

- Considering that our dataset comprises 123 columns, we will need to narrow down the selection to only those attributes that are meaningful in predicting whether a date will result in a match or not.

- We need to select the significant features based on common knowledge and quantified feature importance results:
    - Common Knowledge: manually dropping columns by looking at each columns
    - Quantified: use feature selection model for each model we perform on the dataset to select optimal number of features

Data Preprocessing: Feature Selection and Missing value

The top two columns with the most missing data are:

1. expected_num_interested_in_me : Out of the 20 people you will meet, how many do you expect will be interested in dating you?
2. expected_num_matches: How many matches do you expect to get?

```
df2.isnull().sum().sort_values(ascending=False).head(20)

expected_num_interested_in_me    6578
expected_num_matches             1173
```

Decided to drop the two columns based on following reasons:

1. These two variables are not essential for our predictive analysis intuitively
2. Substituting the missing values with the median or mean could lead to misleading results, and dropping all of the null values was not a feasible solution since we only have 8378 rows.

Data Preprocessing: Feature Selection and Missing value

- Dropped the rows with missing values with 5617 rows left, since filling out na values with mean or median can potentially introduce bias and deviate from the true values,

```
df2.dropna().shape
```

```
(5617, 121)
```

and the distribution of variable gender is very balanced

```
df2.dropna()['gender'].value_counts()
```

```
b'male'      2813
b'female'    2804
Name: gender, dtype: int64
```

- Dropped the first column as it contains only a single unique value

```
[len(df[x].unique()) for x in df2.columns][0:10]
```

```
[1, 21, 2, 25, 25, 35, 4, 6, 6, 2]
```

Data Preprocessing: Multicollinearity

- We found that for all columns that starts with "d_", they represent the binned range of previous numeric variables.
    - E.g., column 'd_importance_same_race' would be the binned categorical variable corresponding to the numeric variable 'importance_same_race'.
- These can be seen as duplicates of previous columns, meaning that they are perfectly collinear with their corresponding numeric columns. **Therefore, we have to drop these columns.**

E.g.

Column 'd_importance_same_race' represents the binned categorical variable for the numeric variable 'importance_same_race'.

importance_same_race	d_importance_same_race
2.0	b'[2-5]'
2.0	b'[2-5]'
2.0	b'[2-5]'
2.0	b'[2-5]'
2.0	b'[2-5]'
2.0	b'[2-5]'
2.0	b'[2-5]'
2.0	b'[2-5]'
2.0	b'[2-5]'

Data Preprocessing: Multicollinearity

- We also found that for columns that ends with '_ o' or has '_ o _' in the middle of the column name, they represent the features of the opposite side in the date, i.e., the partner.
- However, we noticed that both sides were included.
- For the features and the corresponding partner features, the distribution were almost identical, meaning they have nearly the same mean and standard deviation.
    - **Therefore, we have to drop these columns**

```
o_columns
```

```
Index(['race_o', 'decision_o', 'age_o', 'attractive_o', 'sinsere_o',
       'intelligence_o', 'funny_o', 'ambitous_o', 'shared_interests_o'],
      dtype='object')
```

E.g.

‘age’ and ‘age_o’ represent the person’s age and age of their partner respectively

'age' has a mean of 26.35 and a standard deviation of 3.5668,

while 'age_o' has a mean of 26.36 and a standard deviation of 3.5636

Data Preprocessing: Multicollinearity

- While checking correlations between x columns and y column, we found the two variables with highest correlation, ‘decision’ and ‘decision_o’ , are very similar to the target variable match
  - If decision is 1, and the partner decision_o is also 1, then match=1
  - Otherwise, match is 0
  - Problematic because of data leakage (y appearing in x columns).

	gender	d_d_age	race	race_o	samerace	met	match
match	-0.003076	-0.049996	0.035791	0.036666	0.008935	0.110964	1.000000
decision_o	-0.112672	-0.032552	0.060678	-0.024796	0.024870	0.053945	0.525731
decision	0.110499	-0.031983	-0.020904	0.060866	0.026511	0.067134	0.523455
like	0.049067	-0.053991	0.030092	0.067674	0.036648	0.139573	0.310860
d_like	0.029440	-0.056300	0.033205	0.062808	0.046689	0.123801	0.282079
...	...	...	...	...	...	...	...
pref_o_sincere	0.119401	-0.004462	0.010470	-0.088415	-0.050910	-0.019138	-0.041998
d_d_age	-0.003614	1.000000	0.020770	0.020819	0.044313	-0.078729	-0.049996
d_age	-0.005290	0.888580	0.018383	0.017926	0.057652	-0.067824	-0.054631
importance_same_race	-0.110823	-0.039816	-0.010684	0.013499	0.104836	0.018016	-0.055283
d_importance_same_race	-0.086495	-0.038364	-0.006854	0.011289	0.088242	0.008590	-0.060720

Data Preprocessing: Imbalanced Data

- The dataset appears to be imbalanced, with a higher number of unmatches than matches.

```
df2['match'].value_counts(normalize=True)
```

```
b'0'    0.823215
b'1'    0.176785
Name: match, dtype: float64
```

- To address the issue of overfitting and misleading accuracy, we used the SMOTE algorithm to randomly oversample the minority class. The sample size for both unmatches and matches in the training dataset is the same now.

```
0    3710
1    3710
Name: match, dtype: int64
```

Modeling

1. Logistic Regression
2. Decision Tree
3. XGBoost
4. Support Vector Machine (attempted)

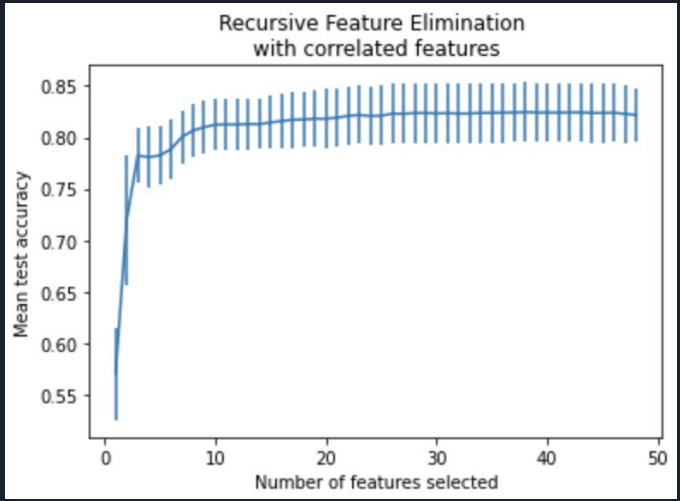
Modelling pipeline

- Feature Selection **-> Recursive Feature Selection (scoring=roc_auc)**

- Hyperparameter Tuning **-> Grid Search**

Logistic Regression - Recursive Feature Selection

- Left is the graph of number of features vs. mean test accuracy
- Optimal number of features is 38
- Below is the ranking of each feature ranked by feature importances. 1's are the features selected




```
rfecv.n_features_
```

```
38
```

```
rfecv.ranking_
```

```
array([ 1,  1,  1, 11,  8,  1,  1,  7,  4,  1,  1,  2,  1,  1,  1,  3,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  6,  9,  1, 10,  1,  1,  1,  1,  1,  5,  1,  1,  1])
```

Logistic Regression - Grid Search and Output

- Training Dataset Classifier Best Accuracy -> **0.825**
- Test Dataset Classifier Accuracy -> **0.710**

```
Tuned Hyperparameters : {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
Traing dataset best Accuracy : 0.8252283113316528
```

```
Accuracy of logistic regression classifier on test set: 0.710
[[662 252]
 [ 74 136]]
              precision    recall  f1-score   support

           0       0.90      0.72      0.80       914
           1       0.35      0.65      0.45       210

    accuracy                           0.71      1124
   macro avg       0.62      0.69      0.63      1124
weighted avg       0.80      0.71      0.74      1124
```

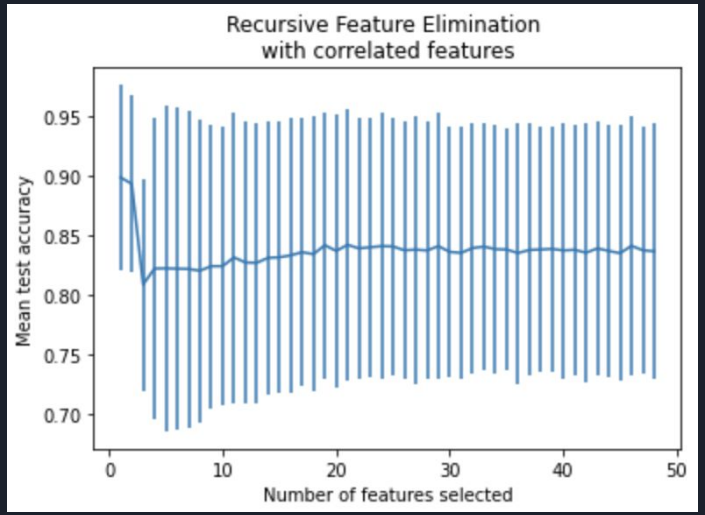
Logistic Regression - Coefficients

- Coefficients contribute positively or negatively to the chance that the two people will match
- Variable “funny” is rating yourself being funny while “funny_partner” is rating your partner on how funny you think he/she is.
- Interestingly, “funny_partner” has a positive coef while “funny” has a negative coef. This means that it’s important for your partner to consider you as funny in a date, rather than considering yourself funny (and that might have the opposite effect)
- Other variables including “attractive_partner” (rate your partner on attractiveness), “interests_correlate” (interests correlation) all have a positive effect on match, which is reasonable
- The variable with the highest positive coef is “met” (whether have met before), which says a lot in a date.
    - Need to get to know each other and become familiar

features	coefficients
samerace	-0.892771
gender	-0.579268
ambition_partner	-0.218209
race	-0.169854
funny	-0.123293
shopping	-0.100814
theater	-0.076560
intelligence_partner	0.125409
shared_interests_partner	0.134840
attractive_partner	0.198231
funny_partner	0.213577
guess_prob_liked	0.224660
like	0.316227
interests_correlate	0.328800
met	0.391104

Decision Tree - Recursive Feature Selection

- Without setting minimum features, our feature selection chose the optimal number of features to be 1. The variable is “like” (whether liked your partner or not)
  - Guess that’s all it matters at the end of the day, according to decision tree
- However, we want to build a model where we can input several features, not just one feature
  - what if in reality the feature 'like' is missing this time
- We need to specify the **minimum number of features** to be chosen



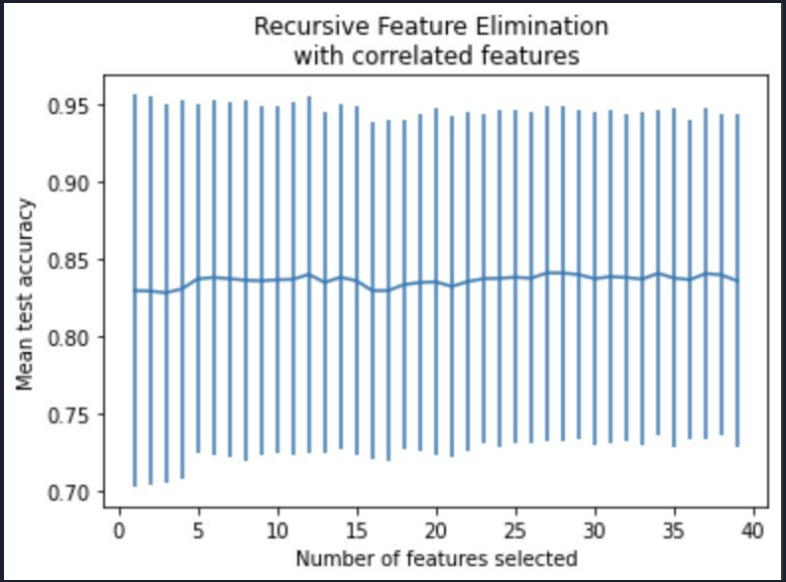

```
rfecv.n_features_
```

```
1
```

```
array([48, 47, 39,  3,  7, 23, 12, 46, 24, 21, 19,  9, 34, 35, 13, 43, 41,
       10, 44, 38,  5, 20, 28,  6, 15,  8, 31, 33, 16, 37, 32, 26, 18, 25,
       30, 42, 11, 40, 17, 14, 36, 29, 27,  2, 45,  1,  4, 22])
```

Decision Tree - Recursive Feature Selection(adjusted)

- Setting the minimum number of features to be 10, we get optimal number of features=36




```
rfecv.n_features_
```

```
36
```

```
rfecv.ranking_
```

```
array([13, 12,  6,  1,  1,  1,  1, 10,  1,  1,  1,  1,  1,  3,  1,  8,  1,
        1,  9,  2,  1,  1,  1,  1,  1,  1,  1,  1,  1,  5,  1,  1,  1,  1,
        4,  1,  1,  1,  1,  1,  7,  1,  1,  1, 11,  1,  1,  1])
```

Decision Tree - Grid Search

Training Dataset Best Accuracy -> **0.883**

Test Dataset Accuracy -> **0.716**

```
Tuned Hyperparameters : {'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 1, 'min_samples_split': 2}
Traing dataset best Accuracy : 0.8833599726825583
```

```
Accuracy of decision tree classifier on test set: 0.716
[[672 242]
 [ 77 133]]
              precision    recall  f1-score   support

           0       0.90      0.74      0.81       914
           1       0.35      0.63      0.45       210

    accuracy                           0.72      1124
   macro avg       0.63      0.68      0.63      1124
weighted avg       0.80      0.72      0.74      1124
```

XGBoost

For XGBoost, nearly all features were selected

Training Dataset Best Accuracy -> **0.971**

Test Dataset Accuracy -> **0.818**

```
rfecv.ranking_
```

```
array([1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3,
       1, 1, 1, 1])
```

```
Tuned Hyperparameters : {'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 140}
Traing dataset best Accuracy : 0.9706831540020779
```

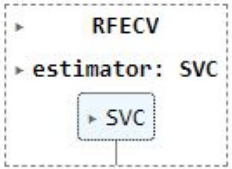
```
Accuracy of XGBoost classifier on test set: 0.818
[[849  65]
 [140  70]]
              precision    recall  f1-score   support

           0       0.86      0.93      0.89       914
           1       0.52      0.33      0.41       210

    accuracy                           0.82      1124
   macro avg       0.69      0.63      0.65      1124
weighted avg       0.79      0.82      0.80      1124
```

Support Vector Machine

```
svm = SVC(kernel='linear')
# recursive feature selection
rfecv_svm = RFECV(
    estimator=svm,
    step=1,
    cv=StratifiedKFold(5),
    scoring='roc_auc'
)
rfecv_svm.fit(X_res, y_res)
```




```
rfecv_svm.ranking_
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1])
```

- We tried to use SVM and the same pipeline
- Specified kernel='linear' in feature selection because the default parameter does not have coefficients or feature importances required by recursive feature selection
- The GridSearch step took way too long to run (8 hours and failed)

Comparing different models

- XGBoost does have the overall highest accuracy (0.818), but our test set is imbalanced, so we cannot just look at the accuracy.
- XGBoost has the lowest recall among all three models (0.33). This means that if we prioritize the true positive rate (out of all the true positives, how many did we predict to be positive), we might want to choose logistic regression model which has the highest recall (0.65)
- However, XGBoost does have the highest precision (0.52), meaning that it has the highest proportion of true positives in all of the predicted positives.
- Overall, our model did well in predicting the 0's, the unmatched, but not very good in predicting the 1's, the matched. Again, this is due to our test target variable is very imbalanced, and accuracy alone in this case can be misleading.

Potential Improvements

- If we had more time, we probably could have successfully ran the SVM model and we would have one more model to compare and choose from
    - We could also try out more parameters (different scoring) and other feature selection models

- If we had more data, we could probably solve the class imbalance problem in the test set and obtain more meaningful results