**6006CEM Assignment: The Data of Dating**

**Contents**

[Task 3a 2](#_Toc184052811)

[Task 3b 5](#_Toc184052812)

[Introduction 5](#_Toc184052813)

[Pre-processing 7](#_Toc184052814)

[Visualisation 9](#_Toc184052815)

[Logistic Regression 12](#_Toc184052816)

[Support Vector Machine 14](#_Toc184052817)

[Multi-layer Perception Classifier 15](#_Toc184052818)

[Conclusion 16](#_Toc184052819)

**Table of Figures**

[Figure 1: number of missing values in dataset 8](#_Toc184127159)

[Figure 2: number of missing values in dataset 8](#_Toc184127160)

[Figure 3: number of missing values in dataset 9](#_Toc184127161)

[Figure 4: number of missing values in dataset 9](#_Toc184127162)

[Figure 5: Pie Charts for Samerace, Met, and Match 9](#_Toc184127163)

[Figure 6: histogram showing distribution of difference in age (d\_age) 10](#_Toc184127164)

[Figure 7: Boxplots for intersts\_correlate, d\_age, and attractive\_o 10](#_Toc184127165)

[Figure 8: histogram showing distribution of difference in age (d\_age) 11](#_Toc184127166)

[Figure 9: Boxplots for interests\_correlate, d\_age, and attractive\_o 11](#_Toc184127167)

[Figure 10: correlation heat map of all variables 12](#_Toc184127168)

[Figure 11: model evaluation metrics 13](#_Toc184127169)

[Figure 12: model parameters 13](#_Toc184127170)

[Figure 13: best model parameters 13](#_Toc184127171)

[Figure 14: model evaluation metrics 13](#_Toc184127172)

[Figure 15: model evaluation metrics 14](#_Toc184127173)

[Figure 16: model parameters 14](#_Toc184127174)

[Figure 17: best model parameters 14](#_Toc184127175)

[Figure 18: model evaluation metrics 15](#_Toc184127176)

[Figure 19: model evaluation metrics 15](#_Toc184127177)

[Figure 20: model parameters 15](#_Toc184127178)

[Figure 21: best model parameters 16](#_Toc184127179)

[Figure 22: model evaluation metrics 16](#_Toc184127180)

**Tables**

[Table 1: Name, Description, and Type of All Variables 7](#_Toc184143062)

[Table 2: Name, Description, and Type of chosen variables 7](#_Toc184143063)

|  |
| --- |
| **Academic Report** |

# **Task 3a**

Sentiment analysis is the process of analysing a piece of text to determine the feeling or emotions behind it. There are many applications for this across business, finance, and politics. For example, some companies use AI to analyse slack/teams messages, employee surveys, and Glassdoor reviews to gauge employee satisfaction. This review will focus on predicting stock price movements. Large movements of specific stocks are dependent on how most investors feel about them. This is why seemingly small controversies can have large impacts on stock price movements; people are often not investing based on how they feel about the company, but how they think others will react. This is further compounded by the fact that often when a share price takes a downturn, more people will sell their shares to lock in profit, further lowering the share price. This isn’t an issue with long-term trading but can have a large impact on high-frequency and day trading.

Financial news articles, social media posts and trends, and analyst opinions can be analysed to give an investor an idea of whether people are generally feeling positive or negative towards a stock. This can help to guide how they manage investments and get ahead of the curve/trend. This data can be very large, unstructured, and include patterns that are almost impossible for humans to work out. Machine learning techniques can be very useful for extracting insights.

However, there are lots of issues of interest. For example, sentiment analysis typically struggles with determining sarcasm and handling noisy data (like lots of irrelevant information).

This review aims to evaluate key studies on sentiment analysis for stock market predictions to find out the machine learning methods used, what data they used, and how accurate the analysis has been for forecasting.

Traditional methods for sentiment analysis include lexicon-based approaches and traditional machine learning models like Naive Bayes and Support Vector Machines (SVM).

Valle-Cruz et al (2022) analysed sentiment from influential financial Twitter accounts, like Bloomberg, CNN Business, and The New York Times using Bing Liu, Sentiment 140, NRC, Affin, and SenticNet lexicons. They used this to evaluate correlations between daily tweet sentiment and stock data for major indices like NASDAQ 100, Dow Jones, S&P 500, FTSE 100 during pandemic periods (June and July 2009 (H1N1) and January and May 2020 (COVID-19)). The results showed that the financial market reacted within 0–10 days for COVID-19 and between 0–15 days for H1N1. It also showed that the more influential Twitter accounts had higher correlations due to their broad audience and more frequent posts. COVID-19 showed stronger and faster effects due to the widespread use of social media in 2020 compared to 2009. This lexicon-based approach was good at capturing general trends, but struggled with the noisiness of twitter data and was not complex enough to deal with issues like sarcasm. Also, this might not be relevant for other financial indices or during non-pandemic periods.

Another study (Shameen, S. 2022) used a naïve bayes to analyse financial news sentiment from financial news articles, looking at Term Frequency (TF), and normalised word counts. They used sentiment embeddings from Loughran and McDonald lexicon, to classify content into positive, neutral, or negative. This approach was very strong, even against some deep learning models. However, it struggled at understanding the relationships between words due to its assumption of feature independence.

The lexicon-based methods proved effective at determining general sentiment/trend in structured data. However, they rely on using dictionaries and can’t adapt to changing industry specific language. In this way, they can struggle with noisy and unstructured data like social media posts due to the prevalence of sarcasm and slang. Naïve bayes also shares some of these issues like not being able to understand the complex relationships between words however was better for structured, labelled datasets. Both of these approaches highlight the need for more advanced methods.

Deep Learning methods for sentiment analysis include advanced Neural Networks to understand complex relationships in text, enabling them to effectively handle unstructured and noisy data.

The previous study (Shameen, S. 2022) compared the naïve bayes against deep learning models like Bidirectional Encoder Representations from Transformers (BERT) and Financial BERT (FinBERT). These approaches are transformer-based models that use attention mechanisms (improves performance by focusing on relevant information) to understand bidirectional context (context from both sides of a word). In this way, these models are able to understand complex and domain-specific language. These models unsurprisingly outperformed the naïve bayes, achieving higher accuracy and F1 scores.

Moore, A, & Rayson, P. (2017) compared using bidirectional long short-term memory (Bi-LSTM) networks against Support Vector Regression (SVR) at predicting sentiment from financial news headlines. The dataset had the headline, the sentiment determined from -1 to 1, and any company names used. The results showed an improvement of 4-6% for the Bi-LSTM with an improvement in handling context. However, it required more computation and struggled when sentences referenced multiple companies.

The transformer-based models proved great at handling unstructured data whereas the Bi-LSTMs were strong at understanding word order and word relation. BERT/FinBERT is great for longer, unstructured, and noisy data, whereas Bi-LSTMs are stronger for shorter datasets.

Hybrid models aim to combine traditional and Deep Learning methods with other data, in this context usually traditional stock price information. The idea is that this combined model will be stronger at predicting stock prices.

Jing, N et al.( 2021) combined Convolutional Neural Networks (CNN’s) with Long Short-Term Memory (LSTM) networks to predict Chinese stock price movements. The dataset included over 880,000 posts from eastmoney.com which is a Chinese stock forum from 2017 to 2019, classified into positive or negative using CNN’s. this was combined with traditional stock data from 6 industries in the Shanghai Stock Exchange (SSE) like moving averages in an LSTM model to predict next day closing prices. This hybrid CNN-LSTM model outperformed the individual models using just sentiment or just traditional stock data, as well as other SVM and traditional LSTM models, with a Mean Absolute Percentage Error (MAPE) of 0.449. However, this was a model trained on Chinese text data, to predict Chinese stock prices. This therefore means it might not necessarily be relevant to other financial markets, or other processed languages.

Picasso, A et al. (2019) used a Feedforward Neural Network (FNN) to combine sentiment from financial news with technical stock data. The dataset had financial news articles and historical stock price data like highs, lows, and volumes for 20 individual NASDAQ stocks between March 2017 and June 2018. They used sentiment embeddings from Loughran & McDonald, and AffectiveSpace lexicons and found that the combined FNN had better predictions than just the sentiment analysis or technical models alone. Accuracy especially was much better than for the individual models. They did a trading simulation which had an annualised return of 85.2% with Loughran and McDonald, and a similar return for AffectiveSpace. However, these results are focused on just the NASDAQ 100 index and may not be relevant to other markets or indices.

The success of these hybrid models shows the strength of combining sentiment analysis with numerical indicators for financial predictions. The CNN-LSTM was especially good at tracking how the sentiment changes might influence stock prices as the CNN’s identified important patterns in the text, whereas the LSTM processed this sequence over time. The FNN showed the potential for real-world applications through its use in high-frequency trading simulations. These hybrid models can help to overcome the issues with just sentiment analysis models or just numerical models. However, they often rely on large, detailed, labelled datasets and require a lot of computing power.

These studies highlight the pros and cons of using traditional, Deep Learning, and hybrid methods for sentiment analysis for financial stock price predictions. Traditional methods like lexicon-based approaches (Valle-Cruz et al. 2022) and naïve bayes (Shameen, S. 2022) were effective at identifying general trends from structured data like news articles. However, they struggled more with unstructured data, sarcasm, and slang like social media posts. Deep learning models showed a significant improvement. Transformer based models like BERT and FinBERT (Shameen, S. 2022) were great at understanding the complex relationships between phrases, and Bi-LSTM’s (Moore, A, & Rayson, P. 2017) were great at understanding complicated contextual information but required large datasets and lots of computation. Hybrid models delivered the most accurate stock price predictions. The CNN-LSTM model (Jing et al., 2021) was strong at tracking accurate stock price movement, while the FNN model (Picasso et al., 2019) showed real-world applications for high-frequency trading. However, these models are reliant on large, detailed, labelled datasets and require a lot of computing power. In the future, it would be a good idea to investigate Machine Learning approaches that keep the same complexity and effectiveness of the model, but without some of these issues.

In conclusion, the best approach depends on the size and complexity of your dataset, as well as your access to computational resources, however using a hybrid approach is likely your best option.

# 

# **Task 3b**

# **Introduction**

During speed dating events, people meet potential partners in a formal environment in the hopes of getting to know and date them. This dataset is from experimental speed dating events between 2002 and 2004. It includes subjective information about how the participants rated each other on different attributes, as well as objective information about their demographics, and whether they have met before. If we can train a model to predict whether a couple will match, we can use it to find the most likely compatible partners for future speed dating events. This is a binary classification problem as we are predicting whether they match or not. The model will be trained on labelled data, so this is supervised learning.

Before starting analysis, it was of interest to see what analysis some other people had done on the same dataset, specifically focusing on preprocessing, pre-analysis and prediction models created. There were a few analyses on Kaggle.com that were useful in these aspects.

@AMR7AC. (2024) imputed missing values with either the mean, median, or modes of those variables, and dropped all duplicates. Initial plots done involved distribution plots and bar charts for categorical variables ‘match’ and ‘gender’, as well as a correlation heat map of all numerical variables.

@Jason\_Chiahs. (2024) picked a subset of the data and filled their missing values with 0 for all numerical variables before plotting a correlation heat map. Pie charts were plotted to show gender distributions, numerical features were scaled with RobustScaler and categorical features encoded with OneHotEncoder, before performing PCA cluster analysis.

@Ushnish Bhowmik. (2024) dropped variables, then imputed missing values with SimpleImputer and scaled with MinMaxScaler before encoding categorical variables with OneHotEncoder. A Keras neural network predicting probabilities was fit with a training-validation-test set split of 72:18:10 before being converted to binary using a threshold of 0.5 and used accuracy as an evaluation metric.

Fisman, R et al. (2006) explores the difference between how men and women evaluate partners. Data was organised by participant and wave to control for group effects and individual variables were modelled against outcome. Linear regression models were used to determine relationships between participant ratings and decision while fixed-effects regression models were used to control for other factors.

|  |  |  |
| --- | --- | --- |
| **Variable(s)** | **Description** | **Type** |
| **gender** | Gender of self | binary categorical [male or female] |
| **age/age\_o** | Age of self/ age of partner | numeric, years |
| **d\_age** | Difference in age | numeric, years |
| **race/race\_o** | Race of self/ race of partner | categorical, 6 levels |
| **samerace** | Whether the two persons have the same race or not | binary categorical [yes or no] |
| **importance\_same\_race, importance\_same\_religion** | How important is it that partner is of same race/religion? | numeric [1 to 10] |
| **field** | Field of study | categorical, 260 levels |
| **pref\_o\_attractive, pref\_o\_sinsere, pref\_o\_intelligence, pref\_o\_funny, pref\_o\_ambitious, pref\_o\_shared\_interests** | How important does the partner rate each attribute? | numeric [1 to 10] |
| **attractive\_o, sincere\_o, intelligence\_o, funny\_o, ambitous\_o, shared\_interests\_o** | Rating by partner (about participant) at night of event on each attribute | numeric [1 to 10] |
| **attractive\_important, sincere\_important, intellicence\_important, funny\_important, ambtition\_important, shared\_interests\_important** | What do you look for in a partner? | numeric, sum to 100 |
| **attractive, sincere, intelligence, funny, ambition**: | Rate yourself on each attribute | numeric [1 to 10] |
| **attractive\_partner, sincere\_partner, intelligence\_partner, funny\_partner, ambition\_partner, shared\_interests\_partner**: | Rate your partner on each attribute | numeric [1 to 10] |
| **sports, exercise, dining, museums, art, hiking, gaming, clubbing, reading, tv, theatre, movies, concerts, music, shopping, yoga** | Rating of your own interests | numeric [1 to 10] |
| **interests\_correlate** | Correlation between participant’s and partner’s ratings of interests | numeric [-1 to 1] |
| **expected\_happy\_with\_sd\_people** | How happy do you expect to be with the people you meet during the speed-dating event? | numeric [1 to 10] |
| **expected\_num\_interested\_in\_me** | Out of the 20 people you will meet, how many do you expect will be interested in dating you? | numeric [1 to 20] |
| **expected\_num\_matches** | How many matches do you expect to get? | numeric [1 to 20] |
| **like** | Did you like your partner? | numeric [1 to 10] |
| **guess\_prob\_liked** | How likely do you think it is that your partner likes you? | numeric [1 to 10] |
| **met** | Have you met your partner before? | binary categorical [yes or no] |
| **decision/decision\_o** | Decision at night of event/ Decision of partner at night of event | binary categorical [yes or no] |
| **match** | Did they match? (both decision = yes) | binary categorical [yes or no] |

Table 1: Name, Description, and Type of All Variables

Many of these variables were removed:

Gender, as all pairs were Male and Female so it can’t be an indicator for whether they matched.

Age and age\_o, as we already have difference in age (d\_age), which is a more important age metric.

Race and race\_o, as samerace was thought to be more significant.

Field, as there are 260 levels and it would likely be difficult to find significant links between fields of study.

Attribute importance and preference variables, as they are likely less significant than the partner ratings.

‘guess\_prob\_liked’, as well as Expected and Attribute self-rating variables, as they are very subjective and unlikely to have any real impact on odds of matching.

Individual interest variables, as it seemed more important how much the interests correlated (interests\_correlate).

‘Like’, as it would likely be highly correlated with the rating variables.

decision and decision\_o, as the outcome variable match is more important, and they are directly correlated.

The final chosen variables are:

|  |  |  |
| --- | --- | --- |
| **Variable(s)** | **Description** | **Type** |
| **d\_age** | Difference in age | numeric, years |
| **samerace** | Whether the two persons have the same race or not | binary categorical [yes or no] |
| **attractive\_o, sincere\_o, intelligence\_o, funny\_o, ambitous\_o, shared\_interests\_o** | Rating by partner (about participant) at night of event on each attribute | numeric [1 to 10] |
| **attractive\_partner, sincere\_partner, intelligence\_partner, funny\_partner, ambition\_partner, shared\_interests\_partner**: | Rate your partner on each attribute | numeric [1 to 10] |
| **interests\_correlate** | Correlation between participant’s and partner’s ratings of interests | numeric [-1 to 1] |
| **met** | Have you met your partner before? | binary categorical [yes or no] |
| **match** | Did they match? (both decision = yes) | binary categorical [yes or no] |

Table 2: Name, Description, and Type of chosen variables

# **Pre-processing**

The ‘met’ variable that should be just 1 or 0 for yes or no had a few points with values over 1. We can assume that this is where people have mistakenly put either the number of times they’ve met before or the rating of the person when they met previously so either way we set all of these values to 1 (met previously).

For the numeric variables, non-numeric answers were made NaN. Others were outside the allowed bounds, so were coerced. There were only a few of these values, so were unlikely to skew the data

There were quite a lot of missing values in the dataset:

A screenshot of a computer code

Description automatically generated

Figure 1: number of missing values in dataset

Removing all of these missing values would likely have a large impact on the data. The shared interests and ambition pairs have the highest number of missing values, so these variables have been removed.

A screen shot of a computer code

Description automatically generated

Figure 2: number of missing values in dataset

Compared to the size of the dataset, this is a much more manageable number of missing values. However, the planned models do not allow for any missing values. As most of these are numeric, they can be imputed from the mean or medians. However, the ‘met’ variable is categorical so can’t be imputed, the associated entries were removed.

A screen shot of a computer code

Description automatically generated

Figure 3: number of missing values in dataset

The rest of the missing values were then imputed from the row means.

A screen shot of a computer code

Description automatically generated

Figure 4: number of missing values in dataset

The binary categorical variables have been transformed into dummy variables.

# **Visualisation**

Pie charts for were plotted for categorical variables to show the distributions.

A pie chart with numbers and a blue circle

Description automatically generatedA blue circle with a triangle in the middle

Description automatically generatedA blue circle with orange triangle and black text

Description automatically generated

Figure 5: Pie Charts for Samerace, Met, and Match

The majority of couples are not the same race, a large majority hadn’t met previously, and about 17% of the dates resulted in a match.

All of the rating variables were likely to have similar distributions, so only one has been plotted (attractive\_o). Boxplots for d\_age, and interests\_correlate were also plotted to show outliers and skew as well as a histogram for d\_age to show distribution:

A graph of a graph

Description automatically generated

Figure 6: histogram showing distribution of difference in age (d\_age) pre scaling

A graph with lines and numbers

Description automatically generatedA graph with a line and a line of dots

Description automatically generated with medium confidenceA graph with a line and a rectangle

Description automatically generated

Figure 7: Boxplots for intersts\_correlate, d\_age, and attractive\_o pre scaling

The age variable has a large variability and has many large outliers. It is skewed heavily to the smaller values and not normally distributed. The other 2 variables have slight skews, with small outliers that will likely not affect the outcomes.

The Robust Scaler for d\_age removes the effect of the large outliers, and Standard Scaler on the other numerical values deals with the unique values. However, the scaling used still maintained the same distributions and as they are numeric, this could skew results (especially for large age outliers). Power Transform was used instead.

A graph of a graph

Description automatically generated

Figure 8: histogram showing distribution of difference in age (d\_age) post scaling

A graph with a line and numbers

Description automatically generated with medium confidenceA graph with a rectangle and a line

Description automatically generatedA graph with a line and a rectangle

Description automatically generated

Figure 9: Boxplots for interests\_correlate, d\_age, and attractive\_o post scaling

We can see that the effects of the outliers has been minimised, and they now follow much more normal distributions. To get an idea of correlations between the variables, a correlation heat map was plotted.

A screenshot of a computer

Description automatically generated

Figure 10: correlation heat map of all variables

Though it is interesting to see that the ratings a person gave their partner were all highly correlated (which is what we might expect), we are mostly interested in seeing the correlation of each variable with whether they matched. We can see that the strongest correlation was between whether they matched and the attribute ratings, especially for attractiveness and being funny.

The output variable match is not very balanced, with only around 17% of dates resulting in a match. Therefore, accuracy shouldn’t be used as a measure. Here avoiding false positives (predicting a match wrongly) is not vital, but avoiding false negatives (missing a true match) is important, so we are going to try to maximise recall.

# **Logistic Regression**

Logistic regression gives clear insights into the relationship between features and the target variable and is designed for classification. This is a binary classification problem and logistic regression is a straightforward way to classify outcomes into match (1) or no match (0). After running a baseline logistic regression model, the evaluation metrics are:

A black text on a white background

Description automatically generated

Figure 11: model evaluation metrics

The training score (F1 on training data) indicates that there was high precision and recall during the training. The test score (F1 on test data) is similar to training data suggesting that the model isn’t overfitting. The recall score is very low and suggests that of all actual positives, the model only correctly identified 20% of them. The precision score suggests that of all predicted positives, 59% of them were actually positive which is relatively high. The specificity score is very high, suggesting that of all actual negatives, 97% were correctly identified. The accuracy score suggests that the model correctly predicts the outcome 85% of the time, which is quite high. F1-score is low as it is a mixture of the precision which is relatively high, and the recall which is low.

As only around 17% of dates result in a match, the model is quite biased towards predicting ‘no match’. This is why there is high specificity and accuracy as most instances are 0, but low recall as the model isn’t very good at identifying true positives (1’s). To address this issue, we can either under sample the ‘no match’ class or over sample the ‘match’ class.

The logistic regression model used has the following model parameters:



Figure 12: model parameters

A grid search for the C, solver, class\_weight, and max\_iter parameters has been performed. A smaller C means stronger regularisation which prevents overfitting but a larger C reduces regularisation so the model can fit closer to training data. Optimising C ensures the model doesn’t underfit or overfit. The solver determines which optimisation algorithm is used whereas class weight adjusts the weights of each class. As the data is not very balanced, it was assumed to it would pick ‘balanced’ to give more weight to the no-match class but others were tested to be sure. ‘Max\_iter’ is the number of iterations the optimisation algorithm does, and was picked to ensure the model converges for different regularisation values. A few iterations of the grid search were completed to refine the values to maximise the recall score.

The best model had the parameters:



Figure 13: best hyperparameters

A screen shot of a score

Description automatically generated

Figure 14: model evaluation metrics

The recall has significantly improved suggesting the model identifies 79% of positive cases. The precision and specificity have lowered quite significantly, suggesting more false positives, and less correctly identified true negatives due to overfitting the minority group by using balanced class weights. The accuracy has lowered slightly as the model is focusing on the trade-off between the recall and precision. The F1-Score has significantly improved, as the model is now more suited to the imbalance of our dataset.

# **Support Vector Machine**

Support Vector Machine (SVM) classifier models are supervised Machine Learning Models (MLMs) which work by finding the best boundary (hyperplane) to separate classes. It is good at distinguishing between 2 classes and understanding non-linear relationships in our data.

A baseline SVM classifier (SVC) was created with the following results:

A black text on a white background

Description automatically generated

Figure 15: model evaluation metrics

The training score indicates that there was high precision and recall during the training. The test score is similar to training data suggesting that the model isn’t overfitting. The recall score is still low suggesting that of all actual positives, the model only correctly identified 19% of them. The precision score suggests that of all predicted positives, 73% of them were actually positive. The specificity score suggests that of all actual negatives, 99% were correctly identified. The accuracy score suggests that the model correctly predicts the outcome 86% of the time, which is quite high. Finally, The F1 score is relatively low as the recall is low.

This model has the following model parameters:



Figure 16: model parameters

C and class weight have been tuned for the same reasons as before. ‘Kernel’ defines what type of decision boundary is used and ‘gamma’ defines how influential the individual points are on the decision boundary. ‘Degree’ controls the degree of the polynomial kernel, to see what complexity the model should be to avoid underfitting or overfitting. Finally, ‘tol’ which is the tolerance for the stopping criterion for the optimisation algorithm was tuned ensure the model converges without unnecessary computation. Each of these are important to test however a grid search would end up with 864 combinations to try which would have been very computationally intensive. For this reason, a random search was chosen, again maximising for F1 score.



Figure 17: best hyperparameters

A screen shot of a number

Description automatically generated

Figure 18: model evaluation metrics

The recall has significantly improved suggesting the model identifies 80% of positive cases. The precision and specificity have lowered quite significantly, suggesting more false positives and less correctly identified true negatives probably due to overfitting the minority group by using balanced class weights. The accuracy has lowered slightly as the model is focusing on the trade-off between the recall and precision. The F1 score has significantly improved, as the model is now more suited to the imbalance of our dataset.

# **Multi-layer Perceptron Classifier**

Multi-Layer Perceptron (MLP) classifiers are neural networks which are good at finding non-linear relationships in data. This may be able to understand the complex relationships in the data that the other models could not.

The results for the baseline MLP model are as follows:

A black text on a white background

Description automatically generated

Figure 19: model evaluation metrics

The training score indicates that there was high precision and recall during the training. The test score is lower than that of the training data suggesting that the model is overfitting. A gap of 7.3% is noticeable but not large enough to justify changing the test-train split. The recall score is still low suggesting that of all actual positives, the model only correctly identified 31% of them. The precision score suggests that of all predicted positives, 48% of them were actually positive. The specificity score suggests that of all actual negatives, 94% were correctly identified. The accuracy score suggests that the model correctly predicts the outcome 83% of the time, which is quite high. Finally, The F1 score is relatively low as the recall and precision are low.

However, we can still improve this. Here are the model parameters:

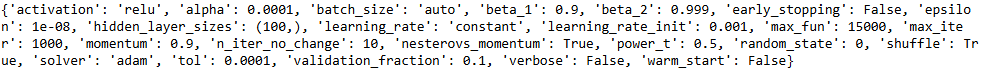


Figure 20: model parameters

‘Hidden\_layer\_sizes’ affects how well the model is able to capture relationships in the data. ‘Activation’ determines the non-linear transformation so the model can learn complex relationships. It also has different impacts on different sizes of hidden layers. ‘Alpha’ controls the regularisation strength, which is important for preventing overfitting, and ‘learning\_rate\_init’ affects the convergence speed.



Figure 21: best hyperparameters

A black text on a white background

Description automatically generated

Figure 22: model evaluation metrics

The recall has slightly improved suggesting the model now identifies 35% of positive cases. The precision and specificity are around the same, suggesting a similar number of false positives and correctly identified true negatives. The accuracy and F1 score are also very similar.

# **Conclusion**

Logistic Regression is good at handling smaller datasets and is computationally efficient. However, it can struggle with non-linear relationships and may underperform with complex decision boundaries. However, there are likely non-linear relationships in the data so SVC and MLP are likely to perform better. SVC is good at handling non-linear relationships well but requires a lot of computation power, so we can only do limited hyperparameter tuning. MLP Classifier can model complex non-linear relationships and handle imbalanced data well so is expected to be the best. However, it is also computationally expensive and requires more tuning compared to the others.

After analysing the results of the different models, the SVC achieved the highest recall meaning it was able to identify 80% of the actual matches correctly. All of the models improved after hyperparameter tuning, with logistic regression reaching 79% and MLP only managing 35%. Similarly, all of the F1-scores increased, suggesting a better balance between the recall and precision. Whilst MLP has the ability to capture non-linear relationships in the data, the presence of overfitting meant that it scored the worst.

For further analysis, we could investigate interaction and polynomial terms to see if there are any deeper relationships between variables. We could also use a different subset of variables, to see if any would lead to increased results. The main issue with the data was the imbalance of the output variable match. We could instead focus on predicting a participant’s likelihood of wanting to match with the ‘decision’ variable, as there is a higher proportion in the minority class.

|  |
| --- |
| **Bibliography** |

Valle-Cruz, D., Fernandez-Cortez, V., López-Chau, A, & Sandoval-Almazán, R. (2022). Does Twitter Affect Stock Market Decisions? Financial Sentiment Analysis During Pandemics: A Comparative Study of the H1N1 and the COVID-19 Periods. <https://link.springer.com/article/10.1007/s12559-021-09819-8>

Shazmeen, S. (2022). Sentiment Analysis of Financial News with Supervised Learning. <https://www.diva-portal.org/smash/record.jsf?dswid=-3228&pid=diva2%3A1636643&c=1&searchType=SIMPLE&language=en&query=shazmeen&af=%5B%5D&aq=%5B%5B%5D%5D&aq2=%5B%5B%5D%5D&aqe=%5B%5D&noOfRows=50&sortOrder=author_sort_asc&sortOrder2=title_sort_asc&onlyFullText=false&sf=all>

Picasso, A., Merello, S., Ma, Y., Oneto, L, & Cambria, E. (2019). Technical analysis and sentiment embeddings for market trend prediction. <https://www.sciencedirect.com/science/article/abs/pii/S0957417419304142>

Moore, A, & Rayson, P. (2017). Lancaster A at SemEval-2017 Task 5: Evaluation metrics matter: predicting sentiment from financial news headlines. [chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://arxiv.org/pdf/1705.00571](chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https:/arxiv.org/pdf/1705.00571)

Jing, N., Wu, Z, & Wang, H. (2021). A hybrid model integrating deep learning with investor sentiment analysis for stock price prediction. <https://www.sciencedirect.com/science/article/abs/pii/S0957417421004607>

Amr7ac. (2024). Speed\_Dating\_Analysis, Kaggle.com. <https://www.kaggle.com/code/amr7ac/speed-dating-analysis>

Jason\_Chiahs. (2024). Speed\_Dating, Kaggle.com. <https://www.kaggle.com/code/jasonchiahs/speed-dating>

Ushnish Bhowmik. (2024). Match Prediction Using Keras and Sklearn, Kaggle.com. <https://www.kaggle.com/code/ushnishbhowmik/match-prediction-using-keras-and-sklearn>

Fisman, R., Iyengar, S., Kamenica, E, & Simonson, I. (2006) Gender Differences in Mate Selection:

Evidence From a Speed Dating Experiment. [chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://www.stat.columbia.edu/~gelman/stuff\_for\_blog/sheena.pdf](chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http:/www.stat.columbia.edu/~gelman/stuff_for_blog/sheena.pdf)

|  |
| --- |
| **Appendix** |

*#importing all packages I need*

**import** **numpy** **as** **np**

**import** **pandas** **as** **pd**

**from** **scipy.io** **import** arff

**import** **matplotlib.pyplot** **as** **plt**

**from** **sklearn.preprocessing** **import** RobustScaler, StandardScaler,LabelEncoder, Normalizer, PowerTransformer

**import** **seaborn** **as** **sns**

**from** **subprocess** **import** check\_output

**from** **sklearn.model\_selection** **import** train\_test\_split,RandomizedSearchCV,GridSearchCV,cross\_val\_score

**from** **sklearn.linear\_model** **import** LogisticRegression

**from** **sklearn.metrics** **import** mean\_squared\_error, r2\_score,accuracy\_score,precision\_score,precision\_recall\_fscore\_support,recall\_score,confusion\_matrix,multilabel\_confusion\_matrix, f1\_score, classification\_report

**from** **scipy.stats** **import** uniform

**from** **sklearn.svm** **import** SVC

**from** **sklearn.neighbors** **import** KNeighborsClassifier

**from** **sklearn.neural\_network** **import** MLPClassifier

**from** **sklearn.datasets** **import** make\_classification

**from** **sklearn.tree** **import** DecisionTreeClassifier

**from** **sklearn.neighbors** **import** KNeighborsClassifier

**from** **sklearn** **import** metrics, tree

*#importing my dataset*

arff\_file = arff.loadarff('data\speeddating.arff')

df1 = pd.DataFrame(arff\_file[0])

*# Set values greater than 1 to 1*

df1.loc[df1["met"] > 1, "met"] = 1

*# Set values outside of 0 and 1 to NaN*

df1.loc[~df1["met"].isin([0, 1]), "met"] = np.nan

fields = [

"attractive\_o", "sinsere\_o", "intelligence\_o",

"funny\_o", "ambitous\_o", "shared\_interests\_o", "attractive\_partner",

"sincere\_partner", "intelligence\_partner", "funny\_partner",

"ambition\_partner", "shared\_interests\_partner"

]

*# Convert numeric fields to numeric*

**for** field **in** fields:

df1[field] = pd.to\_numeric(df1[field], errors='coerce')

*# Clip values to between 0 and 10*

df1[fields] = df1[fields].clip(lower=0, upper=10)

**print**(df1.dtypes)

*#sets values outside of -1 and 1 to NaN*

df1.loc[(df1["interests\_correlate"] < -1) | (df1["interests\_correlate"] > 1), "interests\_correlate"] = np.nan

columns\_to\_keep = [

"gender", "d\_age", "samerace",

"attractive\_o",

"sinsere\_o", "intelligence\_o", "funny\_o", "ambitous\_o",

"shared\_interests\_o", "attractive\_partner", "sincere\_partner",

"intelligence\_partner", "funny\_partner", "ambition\_partner",

"shared\_interests\_partner", "interests\_correlate", "met", "match"

]

*# creates new dataset as subset*

df = df1[columns\_to\_keep]

*#prints number of missing values for each variable*

**print**(df.isna().sum())

*# dropping some variables with large number of missing values*

df = df.drop(columns=["ambitous\_o","shared\_interests\_o", "ambition\_partner","shared\_interests\_partner"])

*#prints number of missing values for each variable*

**print**(df.isna().sum())

*# dropping rows with missing met values*

df = df.dropna(subset=['met'])

*#prints number of missing values for each variable*

**print**(df.isna().sum())

*# if column has missing values and is numeric, replace missing values with mean*

**for** column **in** df.columns:

**if** df[column].isnull().sum() > 0:

**if** df[column].dtype **in** ['float64', 'int64']:

*#df[column].fillna(df[column].median(), inplace=True)*

df[column].fillna(df[column].mean(), inplace=True)

*#prints number of missing values for each variable*

**print**(df.isna().sum())

*# creating dummy variables for categorical*

df = pd.get\_dummies(df, columns=['gender','samerace','met','match'], drop\_first=True)

*# renaming to be more descriptive*

df.rename(columns={"gender\_b'male'": 'gender\_male'}, inplace=True)

df.rename(columns={"samerace\_b'1'": 'samerace\_yes'}, inplace=True)

df.rename(columns={"met\_1.0": 'met\_yes'}, inplace=True)

df.rename(columns={"match\_b'1'": 'match\_yes'}, inplace=True)

*# pie chart for same race distribution*

counts = df['samerace\_yes'].value\_counts()

labels = ['Yes' **if** x == 1 **else** 'No' **for** x **in** counts.index]

plt.figure(figsize=(6, 6))

plt.pie(counts, labels=labels, autopct='**%1.1f%%**', startangle=90, colors=['skyblue', 'orange'])

plt.title('Same Race Distribution (Yes/No)')

plt.show()

*# pie chart for met before distribution*

counts = df['met\_yes'].value\_counts()

labels = ['Yes' **if** x == 1 **else** 'No' **for** x **in** counts.index]

plt.figure(figsize=(6, 6))

plt.pie(counts, labels=labels, autopct='**%1.1f%%**', startangle=90, colors=['skyblue', 'orange'])

plt.title('Met Before Distribution (Yes/No)')

plt.show()

*# pie chart for match distribution*

counts = df['match\_yes'].value\_counts()

labels = ['Yes' **if** x == 1 **else** 'No' **for** x **in** counts.index]

plt.figure(figsize=(6, 6))

plt.pie(counts, labels=labels, autopct='**%1.1f%%**', startangle=90, colors=['skyblue', 'orange'])

plt.title('Match Outcome Distribution (Yes/No)')

plt.show()

*# histogram of d\_age*

plt.figure(figsize=(6, 6))

plt.hist(df["d\_age"].dropna(),bins=30)

plt.title("histogram of d\_age")

plt.xlabel("d\_age")

plt.ylabel("Value")

plt.show()

*# boxplot of d\_age*

plt.figure(figsize=(4, 6))

plt.boxplot(df["d\_age"].dropna())

plt.title("Boxplot of d\_age")

plt.xlabel("d\_age")

plt.ylabel("Value")

plt.show()

*# boxplot of attractive\_o*

plt.figure(figsize=(4, 6))

plt.boxplot(df["attractive\_o"].dropna())

plt.title("Boxplot of attractive\_o")

plt.xlabel("attractive\_o")

plt.ylabel("Value")

plt.show()

*# boxplot of interests\_correlate*

plt.figure(figsize=(4, 6))

plt.boxplot(df["interests\_correlate"].dropna())

plt.title("boxplot of interests\_correlate")

plt.xlabel("interests\_correlate")

plt.ylabel("Value")

plt.ylim(-1,1)

plt.show()

columns\_to\_standardise = [

"d\_age", "attractive\_o", "sinsere\_o", "intelligence\_o",

"funny\_o", "attractive\_partner",

"sincere\_partner", "intelligence\_partner", "funny\_partner",

"interests\_correlate"

]

*#robust\_scaler = RobustScaler()*

*#normaliser = Normalizer()*

*#standard\_scaler = StandardScaler()*

*#df["d\_age"] = robust\_scaler.fit\_transform(df[["d\_age"]])*

*#df[columns\_to\_standardise] = standard\_scaler.fit\_transform(df[columns\_to\_standardise])*

*# power transforming numerical variables*

power\_transform = PowerTransformer(method="yeo-johnson")

df[columns\_to\_standardise] = power\_transform.fit\_transform(df[columns\_to\_standardise])

*# histogram of d\_age*

plt.figure(figsize=(6, 6))

plt.hist(df["d\_age"].dropna(),bins=15)

plt.title("histogram of d\_age")

plt.xlabel("d\_age")

plt.ylabel("Value")

plt.show()

*# boxplot of d\_age*

plt.figure(figsize=(4, 6))

plt.boxplot(df["d\_age"].dropna())

plt.title("Boxplot of d\_age")

plt.xlabel("d\_age")

plt.ylabel("Value")

plt.show()

*# boxplot of attractive\_o*

plt.figure(figsize=(4, 6))

plt.boxplot(df["attractive\_o"].dropna())

plt.title("Boxplot of attractive\_o")

plt.xlabel("attractive\_o")

plt.ylabel("Value")

plt.show()

*# boxplot of interests\_correlate*

plt.figure(figsize=(4, 6))

plt.boxplot(df["interests\_correlate"].dropna())

plt.title("boxplot of interests\_correlate")

plt.xlabel("interests\_correlate")

plt.ylabel("Value")

plt.ylim(-1,1)

plt.show()

*# check for correct numbers of unique values*

**print**(df.nunique())

*# correlation heat map plot for all variables*

corr = df.corr()

plt.figure(figsize=(15,15))

sns.heatmap(corr,vmax=1,linewidth=.01, square = True, annot = True,cmap='YlGnBu',linecolor ='black')

plt.title('Correlation between features')

*# creating the test-train split*

x=df.iloc[1:,:-1].values

y=df.iloc[1:,-1].values

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.2,random\_state=0)

*# fitting baseline logistic regression model to data*

model=LogisticRegression(max\_iter=50)

model.fit(x\_train,y\_train)

train\_score = model.score(x\_train, y\_train)

test\_score = model.score(x\_test, y\_test)

**print**("Training score: {:.3f}".format(train\_score))

**print**("Test score: {:.3f}".format(test\_score))

*# creating confusion matrix to evaluate model scores*

y\_pred = model.predict(x\_test)

cm = confusion\_matrix(y\_test, y\_pred)

tn = cm[0, 0]

tp = cm[1, 1]

fn = cm[1, 0]

fp = cm[0, 1]

recall\_score = (tp / (tp + fn))

precision\_score = (tp / (tp + fp))

**print**('precision score is {:.2f} %'.format(precision\_score\*100))

**print**('specificity score is {:.2f} %'.format(tn / ((tn + fp))\*100))

**print**('accuracy score is {:.2f} %'.format((tp+tn) / ((tn+tp+fn+fp))\*100))

f1\_1 = 2\*recall\_score\*precision\_score

f1\_2 = recall\_score+precision\_score

f1 = f1\_1/f1\_2

**print**('F1 score is {:.2f} %'.format(f1\*100))

**print**('recall score is {:.2f} %'.format(recall\_score\*100))

*# getting the model parameters to see which we could tune for grid search*

**print**(model.get\_params())

*# performing a grid search*

param\_grid = {

'C': [1.5,2,2.5],

'solver': ['lbfgs', 'saga', 'liblinear'],

'class\_weight': [None, 'balanced'],

'max\_iter': [22, 23, 24, 50],

}

grid\_search = GridSearchCV(model, param\_grid, cv=5, scoring='recall')

grid\_search.fit(x\_train, y\_train)

**print**(grid\_search.best\_params\_)

score = grid\_search.score(x\_train, y\_train)

**print**('Training score is :{:.3f}'.format(score))

score = grid\_search.score(x\_test,y\_test)

**print**('Test score is :{:.3f}'.format(score))

y\_pred = grid\_search.predict(x\_test)

*# creating confusion matrix to evaluate model scores*

cm = confusion\_matrix(y\_test, y\_pred)

tn = cm[0, 0]

tp = cm[1, 1]

fn = cm[1, 0]

fp = cm[0, 1]

recall\_score = (tp / (tp + fn))

precision\_score = (tp / (tp + fp))

**print**('precision score is {:.2f} %'.format(precision\_score\*100))

**print**('specificity score is {:.2f} %'.format(tn / ((tn + fp))\*100))

**print**('accuracy score is {:.2f} %'.format((tp+tn) / ((tn+tp+fn+fp))\*100))

f1\_1 = 2\*recall\_score\*precision\_score

f1\_2 = recall\_score+precision\_score

f1 = f1\_1/f1\_2

**print**('F1 score is {:.2f} %'.format(f1\*100))

**print**('recall score is {:.2f} %'.format(recall\_score\*100))

*# fitting baseline SVM classifier model to data*

model = SVC()

model.fit(x\_train, y\_train)

y\_pred = model.predict(x\_test)

score = model.score(x\_train, y\_train)

**print**('Training score is :{:.3f}'.format(score))

score=model.score(x\_test,y\_test)

**print**('Test score is :{:.3f}'.format(score))

y\_pred = model.predict(x\_test)

*# creating confusion matrix to evaluate model scores*

cm = confusion\_matrix(y\_test, y\_pred)

tn = cm[0, 0]

tp = cm[1, 1]

fn = cm[1, 0]

fp = cm[0, 1]

recall\_score = (tp / (tp + fn))

precision\_score = (tp / (tp + fp))

**print**('precision score is {:.2f} %'.format(precision\_score\*100))

**print**('specificity score is {:.2f} %'.format(tn / ((tn + fp))\*100))

**print**('accuracy score is {:.2f} %'.format((tp+tn) / ((tn+tp+fn+fp))\*100))

f1\_1 = 2\*recall\_score\*precision\_score

f1\_2 = recall\_score+precision\_score

f1 = f1\_1/f1\_2

**print**('F1 score is {:.2f} %'.format(f1\*100))

**print**('recall score is {:.2f} %'.format(recall\_score\*100))

*# getting the model parameters to see which we could tune for random search*

**print**(model.get\_params())

*# performing a random search*

param\_dist = {

'C': (0.5,1,1.5),

'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],

'gamma': ['scale', 'auto', 0.01],

'degree': [ 4, 5, 6],

'class\_weight': [None, 'balanced'],

'tol': [1e-5, 1e-4, 1e-3],

}

randomised\_search = RandomizedSearchCV(estimator=model, param\_distributions=param\_dist, n\_iter=15, cv=3, scoring=('recall'))

randomised\_search.fit(x\_train, y\_train)

best\_params\_rand = randomised\_search.best\_params\_

**print**(f"Best Hyperparameters: {best\_params\_rand}")

score = model.score(x\_train, y\_train)

**print**('Training score is :{:.3f}'.format(score))

score=model.score(x\_test,y\_test)

**print**('Test score is :{:.3f}'.format(score))

best\_model\_rand = randomised\_search.best\_estimator\_

y\_pred\_best\_rand = best\_model\_rand.predict(x\_test)

accuracy\_best\_rand = accuracy\_score(y\_test, y\_pred\_best\_rand)

**print**("Best SVM Accuracy: {:.2f}".format(accuracy\_best\_rand))

y\_pred = randomised\_search.predict(x\_test)

*# creating confusion matrix to evaluate model scores*

cm = confusion\_matrix(y\_test, y\_pred)

tn = cm[0, 0]

tp = cm[1, 1]

fn = cm[1, 0]

fp = cm[0, 1]

recall\_score = (tp / (tp + fn))

precision\_score = (tp / (tp + fp))

**print**('precision score is {:.2f} %'.format(precision\_score\*100))

**print**('specificity score is {:.2f} %'.format(tn / ((tn + fp))\*100))

**print**('accuracy score is {:.2f} %'.format((tp+tn) / ((tn+tp+fn+fp))\*100))

f1\_1 = 2\*recall\_score\*precision\_score

f1\_2 = recall\_score+precision\_score

f1 = f1\_1/f1\_2

**print**('F1 score is {:.2f} %'.format(f1\*100))

**print**('recall score is {:.2f} %'.format(recall\_score\*100))

*# fitting baseline MLPC classifier model to data*

model = MLPClassifier(max\_iter=1000, random\_state=0)

model.fit(x\_train, y\_train)

y\_train\_pred=model.predict(x\_train)

train\_score = model.score(x\_train, y\_train)

test\_score = model.score(x\_test, y\_test)

**print**("Training score: {:.3f}".format(train\_score))

**print**("Test score: {:.3f}".format(test\_score))

y\_pred = model.predict(x\_test)

*# creating confusion matrix to evaluate model scores*

cm = confusion\_matrix(y\_test, y\_pred)

tn = cm[0, 0]

tp = cm[1, 1]

fn = cm[1, 0]

fp = cm[0, 1]

recall\_score = (tp / (tp + fn))

precision\_score = (tp / (tp + fp))

**print**('precision score is {:.2f} %'.format(precision\_score\*100))

**print**('specificity score is {:.2f} %'.format(tn / ((tn + fp))\*100))

**print**('accuracy score is {:.2f} %'.format((tp+tn) / ((tn+tp+fn+fp))\*100))

f1\_1 = 2\*recall\_score\*precision\_score

f1\_2 = recall\_score+precision\_score

f1 = f1\_1/f1\_2

**print**('F1 score is {:.2f} %'.format(f1\*100))

**print**('recall score is {:.2f} %'.format(recall\_score\*100))

*# getting the model parameters to see which we could tune for grid search*

**print**(model.get\_params())

*# performing a grid search*

param\_dist = {

'hidden\_layer\_sizes': [(50,), (100,)],

'activation': ['relu', 'tanh'],

'solver': ['adam'],

'alpha': [0.0001, 0.001],

'learning\_rate\_init': [0.001, 0.01],

}

grid\_search = GridSearchCV(model, param\_dist, cv=3, scoring='recall', n\_jobs=-1)

grid\_search.fit(x\_train, y\_train)

**print**(grid\_search.best\_params\_)

train\_score = grid\_search.score(x\_train, y\_train)

test\_score = grid\_search.score(x\_test, y\_test)

**print**("Training score: {:.3f}".format(train\_score))

**print**("Test score: {:.3f}".format(test\_score))

y\_pred = grid\_search.predict(x\_test)

*# creating confusion matrix to evaluate model scores*

cm = confusion\_matrix(y\_test, y\_pred)

tn = cm[0, 0]

tp = cm[1, 1]

fn = cm[1, 0]

fp = cm[0, 1]

recall\_score = (tp / (tp + fn))

precision\_score = (tp / (tp + fp))

**print**('precision score is {:.2f} %'.format(precision\_score\*100))

**print**('specificity score is {:.2f} %'.format(tn / ((tn + fp))\*100))

**print**('accuracy score is {:.2f} %'.format((tp+tn) / ((tn+tp+fn+fp))\*100))

f1\_1 = 2\*recall\_score\*precision\_score

f1\_2 = recall\_score+precision\_score

f1 = f1\_1/f1\_2

**print**('F1 score is {:.2f} %'.format(f1\*100))

**print**('recall score is {:.2f} %'.format(recall\_score\*100))