Machine Learning (CS7CS4)

Final Assignment

Catalin Gheorghiu (22305257)

First Part

Feature Selection

Let us start the report where the analysis started: opening the data sets in a spreadsheet. It is immediately apparent that cleaning up missing values, intractable formats, and more is in order, but first, not all of the data itself seems useful. All of the potential features are organised in columns across data sets, with the rows representing individual properties in the "listings" and individual comments in the "reviews" data set, respectively. Of these columns, many are superfluous or clearly unsuitable for predicting ratings, therefore they ought to be removed outright to facilitate the analysis. A non-exhaustive list of reasons I choose to delete a column in this stage includes:

- The information is only related to the scarping process or page layout, e.g. "scrape_id", "host_picture_url", "source", "calendar_last_scraped", "listing_url"
- The information is obviously unrelated to the property's rating, e.g. "host_name", "latitude/longitude", "first/last_review", "reviewer_name", "date"
- The feature is either a duplicate, included in another column, or contains no information, e.g. "host_listings_count", "host_total_listings_count", and "calculated_host_listings_count" — the latter being subdivided in a further 3 columns — all refer to the same information

Feature Transformation

Starting with the "reviews" data set, I focus squarely on the comment text and the listing id required to connect it to the "listings" data set. In order to process the reviews into features, the goal is to create a TF-IDF framework: the widely used heuristic for identifying informative tokens as simultaneously frequent in one review and infrequent in the overall feature set. Achieving this goal is a three-step process.

First, thousands of reviews are not in English, making it possible for non-English words to show up as features even after removing the very rare tokens. This would lead to an overall decrease in the prediction models' effectiveness, as there would be a comparatively small sample size of non-English reviews with comparatively few tokens capable of gauging the reviewer's sentiment, all while the term frequency of English tokens is skewed downwards by their failure to appear in those thousands of non-English reviews. Consequently, I use the *spaCy* language detection package to identify and keep the English reviews only, removing about 10% of the total number of comments.

Second, I tokenize the reviews' text using the *nltk* package and lemmatize the result using the *WordNetLemmatizer* in order to incorporate knowledge of language structure for the effective merging of similar tokens such as different forms of the same word. Third, I use the *sklearn TfidfVectorizer* to both map the sequence of tokens to a feature vector in bag-of-words style and convert the number of word occurrences to relative frequencies, accomplishing the goal. Along the way, very common tokens — "a", "is", "the" — are removed using the *nltk english stopwords corpus.*

As customary, I select the best level of minimum and maximum document frequency for deciding which features are worth keeping through cross-validation. A baseline linear regression model is applied to the "final" dataset (consisting of the processed reviews being appended to the listings based on listing id) to predict only the rating score; how the "listings" dataset was processed remains to be discussed at the end of this section, after the discussion of the reviews is concluded. Figure 1 reflects the five-fold cross-validation applied to each combination of minimum and maximum document frequency, with the former on the X-axis and the latter coded in line colour. The model achieves lower mean absolute errors (MAE) overall, but the trends and insights mirror the left-hand mean squared error (MSE) panel.

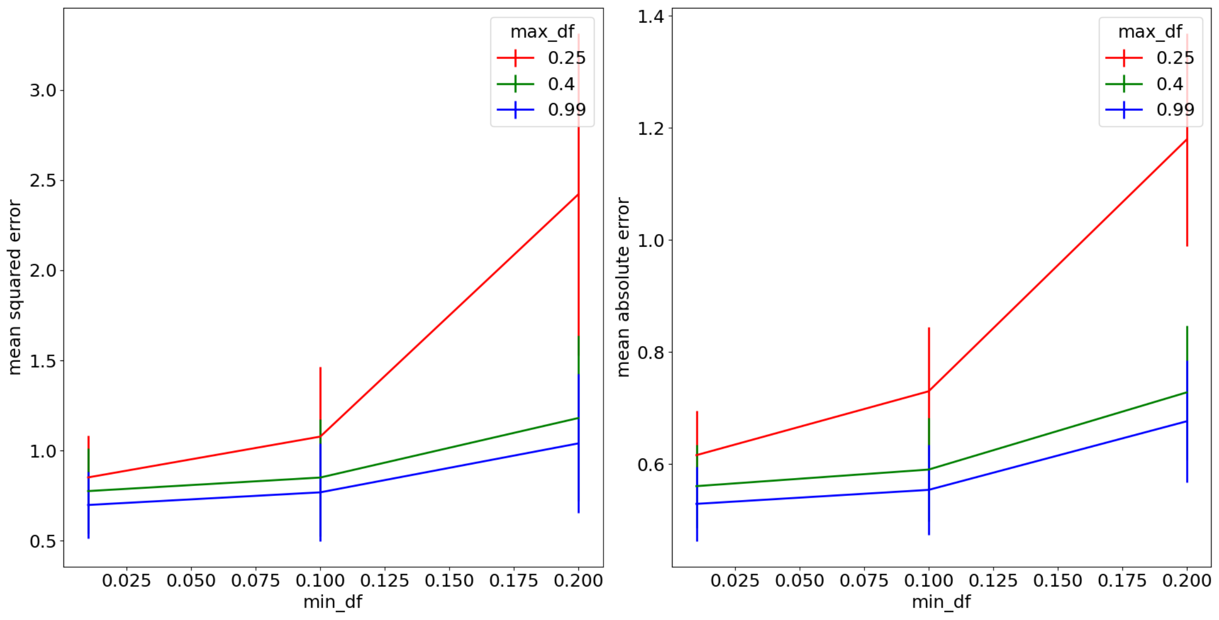

Figure 1: Cross-validation results on the minimum and maximum document frequency to be used in TF-IDF feature transformation, using a baseline model evaluated through mean squared error (left) and mean absolute error (right).

Lower maximum document frequency negatively impacts performance, suggesting that even the most common words have predictive value and leading me to remove this threshold. Similarly, increasing the minimum document frequency intuitively leads to a loss of information as features get removed, but it also increases how generalisable the model is. In order to avoid over-fitting, I set the minimum document frequency at the apparent elbow point of 0.1, meaning that all terms appearing in less than 10% of documents are removed, reducing the feature count from almost 400 to about 30.

Note that several reviews often address the same property. In this case, the TF-IDF values were averaged in order to get one set of features for each listing id. After appending the tokenized review features to the "listings" data set, only listing-based features remain to be transformed. Text features like "description" and "host_about" are converted to the length of the string instead of tokenizing and processing into some bag-of-words variant to keep the number of features low and thus avoid over-fitting. I was ready to use this set of features as a baseline and incrementally expand text columns to increase predictive power, but it will become apparent in future sections why this was not necessary. True-or-false features like "host_is_superhost" have been binarized with "1" meaning "true", and location features like "neighbourhood_cleansed" have also been binarized with "1" meaning that "Dublin" is somewhere in the name; this differentiates between inner-Dublin listings and the surroundings, as guests are often interested in proximity to the city centre. Finally, some features such as "room_type" have been broken down in multiple binary columns, for instance an "is_shared" column and "is_entire_place". Other instances of this strategy are "bathrooms_text" and "host_response_time".

Linear Regression: Ridge & Lasso

Due to its simplicity and robustness, linear regression is a reliable, widely used starting point for continuous value prediction tasks such as this. The *Ridge* and *Lasso* variants — signaling that L2 and L1 regularisation will be employed, respectively — give some fine-tuning control to this otherwise basic method. For each regularisation variant, I heuristically perform five-fold cross-validation with an 80/20 training-test split to determine the best hyperparameter $\alpha = \frac{1}{2C}$. A wide range of C values centered around 1 is used for each of the seven review metrics we are interested in, with a separate model (and consequently separate parameters) used to predict each individual outcome. To assess the performance of the models, I compare them to a dummy regressor that simply predicts the average of the respective outcome over the training set for each listing in the test set.

Figure 2 reflects the cross-validation results. Both regression variants perform better as C increases, but while *Lasso* sees a gradual improvement from 10 to 100, the *Ridge* regression is virtually unchanged for any value of C above 1. At their best, both *Lasso* and *Ridge* outperform the dummy with a roughly 75% lower MSE on all seven review metrics, but the *Ridge* variant still obtains marginally better results than *Lasso*. Therefore, a *Ridge* regression with regularisation hyperparameter $\alpha = \frac{1}{2}$ (corresponding to $C = 1$) is the preferred linear regression setting moving forward. The estimated parameter matrices are too large to show, but it is nevertheless valuable to discuss some of the heavier-weighing values to gain insight into the model; this will be done in the concluding subsection.

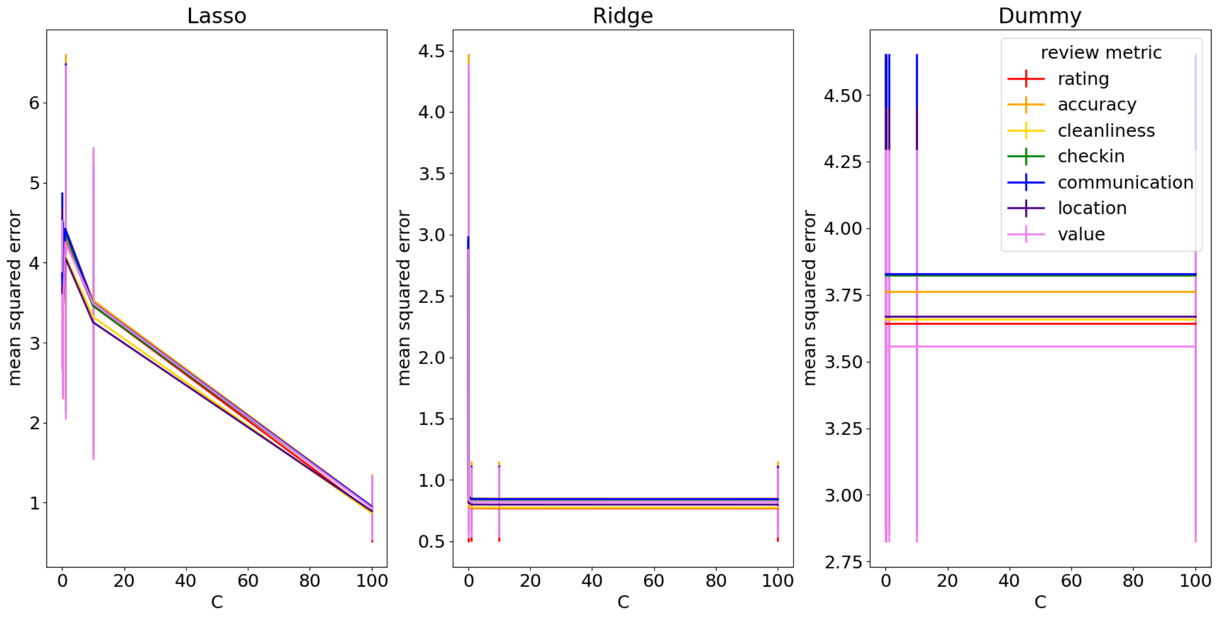

Figure 2: Comparison of *Lasso* and *Ridge* regression under different hyperparameters against a mean-predicting dummy regressor for each review metric.

Multi-Layer Perceptron (MLP)

As we are dealing with over 70 input features and some of them may interact — for instance, a private room with a private bathroom may be more desirable than each part in a void — the potential to identify non-linear relationships in the data is worth exploring. For these reason, as well as the inherent value of contrasting the performance of linear regression to a more black-box artificial neural network, I chose to apply an MLP to the processed dataset. To choose the best number of neurons per hidden layer and the best L2 regularisation hyperparameter $\alpha = \frac{1}{C}$, cross-validation is once again in order.

Figure 3 compares the performances of 20 different hyperparameter combinations for each of the seven review metrics against the same axes to minimise distorted perceptions. For each of these combinations, five-fold cross-validation was used to minimise the impact of noise on the results. Note that, in spite of increasing the maximum number of iterations to 300 and therefore making the algorithm take about an hour to complete, most variants still failed to converge. Nevertheless, it is apparent that the complexity accompanying higher numbers of neurons did not necessarily interact with the failure to converge in order to negatively impact the results. This is clear as the 5 and 100-neuron variants seem to be the best performers, with no MSE values above 500 as long as C was greater than 1. Between these two, the hidden layer size of 5 achieves lower MSE values on average (and across virtually all values of C) and is more consistent between review metrics, making 5 neurons the best choice. Turning our attention to the regularisation hyperparameter in the corresponding top-left panel, the MLP's performance seems to largely stabilise after $C = 200$ for every review metric, with the small note that "communication" score predictions keep improving and "accuracy" score predictions get worse as C increases. Because the overall prediction power seems this similar between $C = 200$ and $C = 500$, by Occam's Razor I choose the smaller C for stronger regularisation and thus reduced over-fitting.

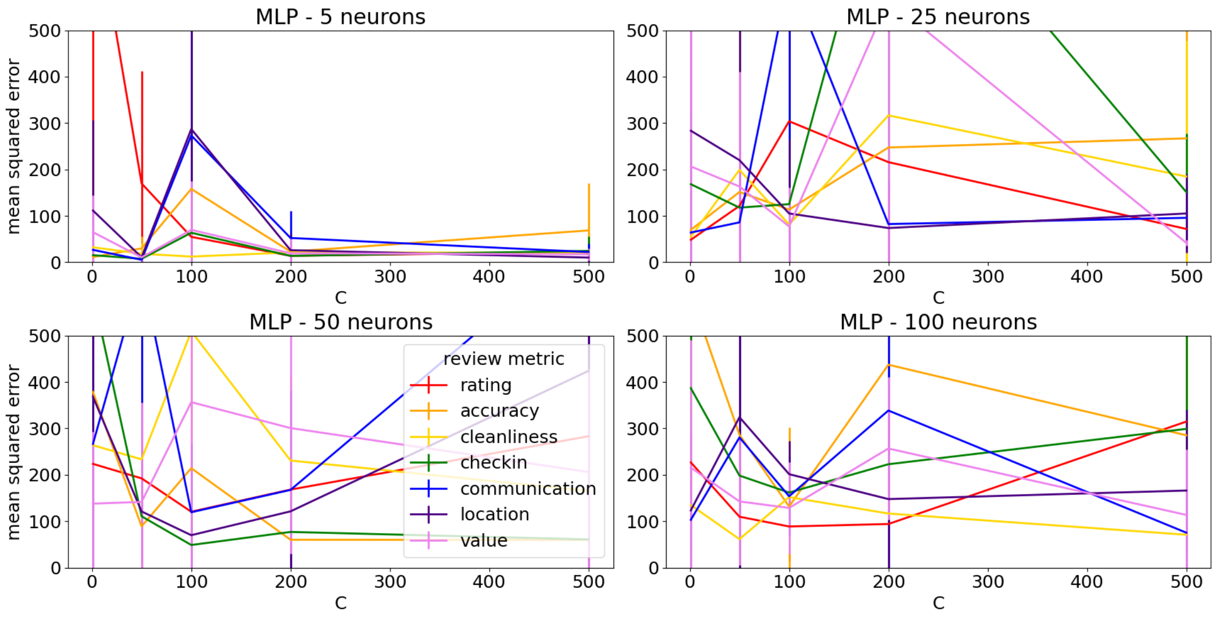

Figure 3: Comparison of different hyperparameter settings for the MLP model. Different panels showcase different hidden layer sizes, while differently coloured lines plot model performance across C values for each review metric.

Discussion & Conclusion

The information in table 1 yields a comparison between the best version of linear regression and MLP respectively that I have found, pit against the mean-predicting dummy regressor. The characteristics of these best versions, mentioned in the previous analysis sections, are given again in the table caption; note that the MLP was given sufficient iterations to converge for this analysis. The *Ridge* regression achieves consistently low MSE and MAE between review scores and always beats the dummy, while the MLP takes a lot more time to only be consistent in MAE and only beat the dummy in two out of seven score predictions for this metric. When it comes to MSE, the MLP dramatically misses the mark on ”value” and ”communication”, but all predictions are a lot worse than the alternatives. Moreover, successive runs of the algorithm yield significantly different values, suggesting that the rather small sample size compared to the usual neural network application or excessive overfitting in spite of the settings tailored to reduce it create too much noise for the MLP to be effective, concluding its analysis.

Table 1: Summarised results of the analysis, comparing the best configuration identified for each method. For linear regression, this is the *Ridge* variant with regularisation $\alpha = \frac{1}{2}$, while for the multi-layer perceptron, it is a variant with 5 neurons per hidden layer with regularisation $\alpha = \frac{1}{200}$.

Target	MSE			MAE			Time (s)	
Review Score	Best_LR	Best_MLP	Dummy	Best_LR	Best_MLP	Dummy	LR	MLP
rating	0.77	15.47	3.64	0.56	2.50	1.52		
accuracy	0.85	15.70	3.76	0.58	1.22	1.55		
cleanliness	0.77	18.69	3.66	0.57	2.37	1.54		
checkin	0.84	31.92	3.82	0.58	2.38	1.56	0.79	67.30
communication	0.85	81.78	3.83	0.58	2.70	1.56		
location	0.80	12.33	3.67	0.57	1.34	1.52		
value	0.82	181.48	3.56	0.57	2.52	1.51		
total	5.70	357.38	25.95	4.01	15.03	10.76		

The impressive performance of the dummy regressor is a reflection of the rather tight clustering of the review scores around 4.5. In these circumstances, it is all the more exciting to conclude that using linear regression to predict review scores is very feasible given the right listing-centric features and a sample of reviews. Table 2 shows the predicted parameters of a selection of impactful features when estimating the ”rating” and ”value” review scores, in order to gain some insight into what drives the rating scores. The entries are ordered by the absolute value of their respective coefficients.

Table 2: Estimated coefficients from the *Ridge* regression of the most impactful features when predicting ”rating” and ”value” review scores, sorted by coefficients' absolute value.

”Rating” Review Score				”Value” Review Score			
Top 5 Review	Coeff.	Top 5 Listing	Coeff.	Top 5 Review	Coeff.	Top 5 Listing	Coeff.
'great'	3.33	'gives_phone'	0.23	'great'	3.26	'gives_phone'	0.30
'lovely'	2.39	'host_has_profile_pic'	0.17	'good'	2.37	'host_response_rate'	-0.17
'place'	2.37	'host_acceptance_rate'	0.14	'place'	2.35	'host_acceptance_rate'	0.13
'home'	2.37	'is_bathroom_shared'	0.10	'nice'	2.28	'host_has_profile_pic'	0.12
'good'	2.35	'host_response_rate'	-0.09	'home'	2.28	'neighbourhood_cleansed'	-0.11

A lot of features are common between the two review scores and the coefficient values are very similar, supporting the intuition that they may be correlated; individuals leaving a high rating will often rate all subcategories highly as well, letting the general impression fill in where targeted thinking should have gone. Another common observation is that all listing features have a substantially lower impact than the review features, suggesting there is little in terms of property features that the reviews do not usually capture.

Looking at the review features, positive adjectives are unsurprisingly most conducive to high ratings. Words referring to the property (”place”, ”home”), while likely having low TF-IDF scores in an AirBnB review database, likely have a positive impact on reviews because the people who reflect on the property itself and even call it ”home” enjoyed their stay.

Finally, in terms of listing features, it is likely that hosts who give their phone and profile picture are more open to establishing an in-person connection with their guests, driving the positive correlation with review scores. Moreover, higher acceptance rates are unsurprisingly appreciated by renters. What is somewhat surprising, however, is that high response rates are the strongest drivers of negative reviews; perhaps hosts with lower response rates are less likely to be bots or corporations, explaining this behaviour. Lastly, the negative impact of ”Dublin” being mentioned in the neighbourhood contradicts the expectation I expressed in the feature transformation section. This may be due to a misalignment between the real neighbourhood and the cleansed variant, or I may have indeed been wrong and people prefer to avoid the inner-Dublin area, perhaps because of the higher prices

Second Part

I

Firstly, if logistic regression is not taken at its best, a poor choice of cost function can cause outliers to skew the decision boundary and thus misclassify the points closer to this boundary. For instance, let squared error be used as the cost function and let $\theta^T x = 0$ fix the decision boundary for a two-class problem with labels -1 and +1. If there is a group of strongly negative outliers without a symmetric positive equivalent, the line $\theta^T x$ that minimises squared error would intersect the X-axis (0) further to the left, causing borderline negative points to be misclassified as positive.

Secondly, even with a better-performing loss function like the standard logistic loss, a difficult use case still arises when there is a class imbalance. Regardless of model configuration, logistic regression would very likely be more biased towards the majority class the greater this imbalance is. This problem only gets exacerbated by classes that are not easily separated linearly.

II

In terms of advantages, kNN is first and foremost transparent due to the lack of a hidden layer and clear formulas to compute weights and distances between points. Furthermore, kNN gives intuitive control over the level of overfitting through how many neighbours are considered, and it also gives control over how impactful further points are is Gaussian weights are employed.

When it comes to disadvantages, kNN is notorious for working poorly with large volumes of data as every point requires a look at the entire dataset, while operating with parallelised mini-batches and the bottleneck-like forward propagation process allow MLP to deal with more data effectively. Moreover, kNN does not offer insights into the relationships within the data, whereas MLP leaves behind a complete set of weights and parameters.

III

Models generalise poorly when they are over-fit to the training data, incorporating random noise in the predictions alongside the true correlations. Splitting the data into a validation set and a training set only once leaves the model vulnerable to the influence of noise in both sets: in a classification problem for instance, it is possible that the randomly selected points in the test set are unbalanced in favour of one class, perhaps leaving that class underrepresented in the training set.

By resampling several times, k-fold cross-validation ensures every point eventually belongs to the test set and is used to train the model multiple times, smoothing out the noise by averaging prediction parameters across folds. If predictions vary significantly between splits, it is likely that the data is very noisy and the model may generalise poorly since meaningful patterns are difficult to pin down.

In choosing the value of k, two opposing influences must be considered: the training set ought to be large in order to ensure the underlying patterns are found, encouraging larger values of k, but the test set should also be large to reduce the impact of noise in a single fold and large values of k imply several more computations from re-fitting the model, encouraging smaller values of k. 5 to 10 splits are considered a middle ground of the two influences, making them good values to avoid both under and over-fitting.

IV

Multi-step predictions use lagged outputs as features. Given training data for arbitrary number q of periods x_{k-q}, x_{k-q+1}, $\dots$ x_{k-1} with $k > q$, if one were to simultaneously predict y_k, y_{k+1} and y_{k+2}, the parameters for the first q training points would be used to predict y_k. This output becomes lagged when it is used alongside x_{k-q+1}, x_{k-q+2}, $\dots$ x_{k-1} to predict y_{k+1}. Note that only y_k gets a new parameter predicion, as the other $q-1$ terms keep their old estimated parameters. The more steps ahead are predicted in one go, the more such lagged outputs are used; this may be dangerous as compounding errors can decrease prediction accuracy for later moments.

Appendix - Python Code

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import pandas as pd
import numpy as np
import sklearn
import matplotlib.pyplot as plt
from spacy_langdetect import LanguageDetector
from spacy.language import Language
import spacy
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import mean_absolute_error as MAE
from sklearn.dummy import DummyRegressor
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neural_network import MLPRegressor
import time

plt.rc('font', size=18); plt.rcParams['figure.constrained_layout.use'] = True

#read csvs and show shape
listing_df = pd.read_csv("listings.csv")
reviews_df = pd.read_csv("reviews.csv")
print(listing_df.head())
print("-----------")
print(reviews_df.head())
print("-----------")

#drop useless columns and missing values
listing_df = listing_df.drop(columns=['listing_url', 'scrape_id', 'last_scraped',
                                      'source', 'name', 'picture_url',
                                      'host_id', 'host_url', 'host_name',
                                      'host_since', 'host_thumbnail_url',
                                      'host_picture_url', 'host_neighbourhood',
                                      'host_listings_count', 'neighbourhood',
                                      'neighbourhood_group_cleansed', 'latitude',
                                      'longitude', 'property_type',
                                      'bathrooms', 'bedrooms', 'beds',
                                      'minimum_nights', 'maximum_nights',
                                      'minimum_minimum_nights', 'maximum_minimum_nights',
                                      'minimum_maximum_nights', 'maximum_maximum_nights',
                                      'calendar_updated', 'has_availability',
                                      'availability_30', 'availability_60',
                                      'availability_90', 'availability_365',
                                      'calendar_last_scraped', 'number_of_reviews_l30d',
                                      'first_review', 'last_review',
                                      'license', 'calculated_host_listings_count',
                                      'calculated_host_listings_count_entire_homes',
                                      'calculated_host_listings_count_private_rooms',
                                      'calculated_host_listings_count_shared_rooms'])
reviews_df = reviews_df.drop(columns=['id', 'date', 'reviewer_id', 'reviewer_name'])
print('Dimensions before dropping the rows with missing values:')
print(listing_df.shape)
print("-----------")
print(reviews_df.shape)
print("-----------")
print('Dimensions after dropping the REVIEWS with missing values:')
reviews_df.dropna(inplace=True)
reviews_df.reset_index(drop=True, inplace=True)
print(reviews_df.shape)
print("-----------")

#cleaning listings
listing_df['description'] = listing_df['description'].apply(str)
listing_df['description'] = listing_df['description'].apply(len)
listing_df['neighborhood_overview'] = listing_df['neighborhood_overview'].apply(str)
listing_df['neighborhood_overview'] = listing_df['neighborhood_overview'].apply(len)

def binarize(key, column):
    column = column.apply(str)
    for i in range(len(column)):
        if key in column[i]:
            column[i] = 1
        else:
            column[i] = 0
    return column
listing_df['host_location'] = binarize('Dublin', listing_df['host_location'])

listing_df['host_about'] = listing_df['host_about'].apply(str)
listing_df['host_about'] = listing_df['host_about'].apply(len)

for i in range(len(listing_df['host_response_time'])):
    if listing_df['host_response_time'][i] == 'within an hour':
        listing_df['host_response_time'].at[i] = 1
    elif listing_df['host_response_time'][i] == 'within a few hours':
        listing_df['host_response_time'].at[i] = 5
    elif listing_df['host_response_time'][i] == 'within a day':
        listing_df['host_response_time'].at[i] = 20
    elif listing_df['host_response_time'][i] == 'a few days or more':
        listing_df['host_response_time'].at[i] = 50
    else: listing_df['host_response_time'].at[i] = 10000

listing_df['host_response_rate'] = \
    listing_df['host_response_rate'].str.rstrip('%').astype('float') / 100.0
listing_df['host_acceptance_rate'] = \
    listing_df['host_acceptance_rate'].str.rstrip('%').astype('float') / 100.0

listing_df['host_is_superhost'] = binarize('t', listing_df['host_is_superhost'])

listing_df['gives_email'] = 0
listing_df['gives_phone'] = 0
listing_df['gives_work_email'] = 0
for i in range(len(listing_df['host_verifications'])):
    if "'email'" in listing_df['host_verifications'][i]: listing_df['gives_email'].at[i] = 1
    if "'phone'" in listing_df['host_verifications'][i]: listing_df['gives_phone'].at[i] = 1
    if "'work_email'" in listing_df['host_verifications'][i]: listing_df['gives_work_email'].at[i] = 1
listing_df.drop(columns=['host_verifications'], inplace=True)

listing_df['host_has_profile_pic'] = binarize('t', listing_df['host_has_profile_pic'])

listing_df['host_identity_verified'] = binarize('t', listing_df['host_identity_verified'])

listing_df['neighbourhood_cleansed'] = binarize('Dublin', listing_df['neighbourhood_cleansed'])

listing_df['is_entire_place'] = 0
listing_df['is_shared'] = 0
for i in range(len(listing_df['room_type'])):
    if 'entire' in listing_df['room_type'][i]: listing_df['room_type'].at[i] = 1
    if 'shared' in listing_df['room_type'][i]: listing_df['room_type'].at[i] = 1
listing_df.drop(columns=['room_type'], inplace=True)

listing_df['bathrooms_text'] = listing_df['bathrooms_text'].apply(str)
listing_df['is_bathroom_shared'] = 0
listing_df['bathrooms_count'] = 0
for i in range(len(listing_df['bathrooms_text'])):
    if 'hared' in listing_df['bathrooms_text'][i]: listing_df['is_bathroom_shared'].at[i] = 1
    if 'alf' in listing_df['bathrooms_text'][i]: listing_df['bathrooms_count'].at[i] = 0.5
    else:
        listing_df['bathrooms_count'].at[i] = float(listing_df['bathrooms_text'].at[i].split(" ")[0])
listing_df.drop(columns=['bathrooms_text'], inplace=True)

listing_df['amenities'] = listing_df['amenities'].str.split(',').apply(len)

listing_df['price'] = listing_df['price'].str.lstrip('$')
listing_df.rename(columns={'price': 'property_price'}, inplace=True)

listing_df['instant_bookable'] = binarize('t', listing_df['instant_bookable'])

listing_df = listing_df.iloc[:, [0,20,21,22,23,24,25,26,1,2,3,4,5,6,7,8,9,10,
                                 11,12,13,14,15,16,17,18,19,27,28,29,30,
                                 31,32,33,34,35]]

#remove non-english reviews

def get_lang_detector(nlp, name):
    return LanguageDetector()
```

```
nlp = spacy.load('en_core_web_sm')
Language.factory("language_detector", func=get_lang_detector)
nlp.add_pipe('language_detector', last=True)
for i in range(len(reviews_df)):
    print(i)
    doc = nlp(reviews_df.at[i, 'comments'])
    detect = doc._.language
    if (detect['language'] != 'en'):
        reviews_df = reviews_df.drop(i, inplace=False, axis=0)
reviews_df.reset_index(drop=True, inplace=True)
reviews_df.to_csv('english_reviews.csv', index=False)

#tokenize reviews; previous section can be commented if previously executed
reviews_df = pd.read_csv('english_reviews.csv')
comments = reviews_df.iloc[:,1].to_numpy().tolist()
for i in range(len(comments)):
    print(i)
    comments[i] = word_tokenize(comments[i])
    #comments[i] = [PorterStemmer().stem(token) for token in comments[i]]
    comments[i] = [WordNetLemmatizer().lemmatize(token) for token in comments[i] if token not in set(
        nltk.corpus.stopwords.words('english'))]
    comments[i] = ' '.join(str(t) for t in comments[i])
reviews_df = reviews_df.drop(columns=['comments'])
reviews_df['comments'] = comments
reviews_df.to_csv('clean_reviews.csv', index=False)

#TF-IDF, pruning (min/max_df cross-validation)
colours = ['red', 'green', 'blue']
fig, (ax1, ax2) = plt.subplots(1,2)
reviews_df = pd.read_csv('clean_reviews.csv')
max_df_range = [0.25, 0.4, 0.99]
min_df_range = [0.01, 0.1, 0.2]
for index, max_df in enumerate(max_df_range):
    mean_MSE = []
    mean_MAE = []
    std_MSE = []
    std_MAE = []
    for min_df in min_df_range:
        print("min: " + str(min_df) + " max: " + str(max_df))
        x=reviews_df['comments']
        vectorizer = TfidfVectorizer(stop_words=nltk.corpus.stopwords.words('english'),
                                     max_df=max_df, min_df=min_df)
        X = vectorizer.fit_transform(x.values.astype(str))
        tfidf_df = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())
        appended_df = pd.concat([reviews_df, tfidf_df], axis=1)
        appended_df.drop(columns=['comments'], inplace=True)
        appended_df = appended_df.groupby(appended_df['listing_id'], as_index=False).mean()
        feature_df = pd.merge(listing_df, appended_df.rename(columns={'listing_id':'id'}),
                              on='id', how='left')
        feature_df = feature_df.fillna(0)
        feature_df = feature_df.replace('nan', 0)
        feature_df['property_price'].replace(',', '', regex=True, inplace=True)
        X = feature_df.iloc[:, 8:].to_numpy()
        #using only rating and standard linear regression for cross-validation
        y = feature_df.iloc[:, 1]
        model = LinearRegression()
        temp_MSE = []
        temp_MAE = []
        for train, test in KFold(n_splits=5).split(X):
            model.fit(X[train], y[train])
            ypred = model.predict(X[test])
            temp_MSE.append(MSE(y[test], ypred))
            temp_MAE.append(MAE(y[test], ypred))
            #print(np.column_stack((ypred, y[test])))
            print(model.coef_)
            print(feature_df.iloc[:, 8:].columns)
        mean_MSE.append(np.array(temp_MSE).mean())
        std_MSE.append(np.array(temp_MSE).std())
        mean_MAE.append(np.array(temp_MAE).mean())
        std_MAE.append(np.array(temp_MAE).std())
    ax1.errorbar(min_df_range, mean_MSE, yerr=std_MSE, linewidth=2,
                 color=colours[index], label=max_df)
    ax2.errorbar(min_df_range, mean_MAE, yerr=std_MAE, linewidth=2,
                 color=colours[index], label=max_df)
ax1.legend(loc = "upper right", title="max_df")
ax1.set(xlabel = 'min_df', ylabel = 'mean squared error')
ax2.legend(loc = "upper right", title="max_df")
ax2.set(xlabel = 'min_df', ylabel = 'mean absolute error')

#Definitive feature matrix for analysis
reviews_df = pd.read_csv('clean_reviews.csv')
x=reviews_df['comments']
vectorizer = TfidfVectorizer(stop_words=nltk.corpus.stopwords.words('english'), min_df=0.1)
X = vectorizer.fit_transform(x.values.astype(str))
tfidf_df = pd.DataFrame(X.toarray(), columns=vectorizer.get_feature_names_out())
appended_df = pd.concat([reviews_df, tfidf_df], axis=1)
appended_df.drop(columns=['comments'], inplace=True)
appended_df = appended_df.groupby(appended_df['listing_id'], as_index=False).mean()
feature_df = pd.merge(listing_df,
                      appended_df.rename(columns={'listing_id':'id'}), on='id', how='left')
feature_df = feature_df.fillna(0)
feature_df = feature_df.replace('nan', 0)
feature_df['property_price'].replace(',', '', regex=True, inplace=True)
feature_df.to_csv('full_features.csv', index=False)

#Ridge, LASSO and dummy under different C, fixed min/max_df
colours = ['red', 'orange', 'gold', 'green', 'blue', 'indigo', 'violet']
X = feature_df.iloc[:, 8:].to_numpy()
fig, (ax1, ax2, ax3) = plt.subplots(1,3)
C_range = [0.001, 0.1, 1, 10, 100]
for i in range(1,8):
    y = feature_df.iloc[:, i]
    mean_MSE_L = []
    std_MSE_L = []
    mean_MSE_R = []
    std_MSE_R = []
    mean_MSE_D = []
    std_MSE_D = []
    for C in C_range:
        model_L = Lasso(alpha=1 / (2 * C))
        model_R = Ridge(alpha=1 / (2 * C))
        model_D = DummyRegressor(strategy='mean')
        temp_MSE_L = []
        temp_MSE_R = []
        temp_MSE_D = []
        for train, test in KFold(n_splits=5).split(X):
            model_L.fit(X[train], y[train])
            ypred = model_L.predict(X[test])
            temp_MSE_L.append(MSE(y[test], ypred))
            model_R.fit(X[train], y[train])
            ypred = model_R.predict(X[test])
            temp_MSE_R.append(MSE(y[test], ypred))
            model_D.fit(X[train], y[train])
            ypred = model_D.predict(X[test])
            temp_MSE_D.append(MSE(y[test], ypred))
        mean_MSE_L.append(np.array(temp_MSE_L).mean())
        std_MSE_L.append(np.array(temp_MSE_L).std())
        mean_MSE_R.append(np.array(temp_MSE_R).mean())
        std_MSE_R.append(np.array(temp_MSE_R).std())
        mean_MSE_D.append(np.array(temp_MSE_D).mean())
        std_MSE_D.append(np.array(temp_MSE_D).std())
    ax1.errorbar(C_range, mean_MSE_L, yerr=std_MSE_L, linewidth=2, color=colours[i-1])
    ax2.errorbar(C_range, mean_MSE_R, yerr=std_MSE_R, linewidth=2, color=colours[i-1])
    ax3.errorbar(C_range, mean_MSE_D, yerr=std_MSE_D, linewidth=2, color=colours[i - 1],
                 label=feature_df.columns[i].split('_')[2])
ax1.set(xlabel = 'C', ylabel = 'mean squared error', title = 'Lasso')
ax2.set(xlabel = 'C', ylabel = 'mean squared error', title = 'Ridge')
ax3.legend(loc = "upper right", title="review metric")
ax3.set(xlabel = 'C', ylabel = 'mean squared error', title = 'Dummy')

#MLP under different numbers of hidden layers
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2,2)
C_range = [1, 50, 100, 200, 500]
for i in range(1,8):
    y = feature_df.iloc[:, i]
    mean_MSE_M5 = []
    std_MSE_M5 = []
    mean_MSE_M25 = []
    std_MSE_M25 = []
    mean_MSE_M50 = []
    std_MSE_M50 = []
    mean_MSE_M100 = []
    std_MSE_M100 = []
    for C in C_range:
        model_M5 = MLPRegressor(alpha=1 /C, hidden_layer_sizes=(5), max_iter=300)
        model_M25 = MLPRegressor(alpha=1 /C, hidden_layer_sizes=(25), max_iter=300)
        model_M50 = MLPRegressor(alpha=1 /C, hidden_layer_sizes=(50), max_iter=300)
        model_M100 = MLPRegressor(alpha=1 /C, hidden_layer_sizes=(100), max_iter=300)
        temp_MSE_M5 = []
        temp_MSE_M25 = []
        temp_MSE_M50 = []
        temp_MSE_M100 = []
        for train, test in KFold(n_splits=5).split(X):
            model_M5.fit(X[train], y[train])
            ypred = model_M5.predict(X[test])
            temp_MSE_M5.append(MSE(y[test], ypred))
            model_M25.fit(X[train], y[train])
            ypred = model_M25.predict(X[test])
            temp_MSE_M25.append(MSE(y[test], ypred))
            model_M50.fit(X[train], y[train])
            ypred = model_M50.predict(X[test])
            temp_MSE_M50.append(MSE(y[test], ypred))
            model_M100.fit(X[train], y[train])
            ypred = model_M100.predict(X[test])
            temp_MSE_M100.append(MSE(y[test], ypred))
        mean_MSE_M5.append(np.array(temp_MSE_M5).mean())
        std_MSE_M5.append(np.array(temp_MSE_M5).std())
        mean_MSE_M25.append(np.array(temp_MSE_M25).mean())
        std_MSE_M25.append(np.array(temp_MSE_M25).std())
        mean_MSE_M50.append(np.array(temp_MSE_M50).mean())
        std_MSE_M50.append(np.array(temp_MSE_M50).std())
        mean_MSE_M100.append(np.array(temp_MSE_M100).mean())
        std_MSE_M100.append(np.array(temp_MSE_M100).std())
    ax1.errorbar(C_range, mean_MSE_M5, yerr=std_MSE_M5, linewidth=2, color=colours[i - 1])
    ax2.errorbar(C_range, mean_MSE_M25, yerr=std_MSE_M25, linewidth=2, color=colours[i - 1])
    ax3.errorbar(C_range, mean_MSE_M50, yerr=std_MSE_M50, linewidth=2, color=colours[i - 1],
                 label=feature_df.columns[i].split('_')[2])
    ax4.errorbar(C_range, mean_MSE_M100, yerr=std_MSE_M100, linewidth=2, color=colours[i - 1])
ax3.legend(loc = "upper right", title="review metric")
ax1.set(xlabel = 'C', ylabel = 'mean squared error', title = 'MLP - 5 neurons', ylim = [0,500])
ax2.set(xlabel = 'C', title = 'MLP - 25 neurons', ylim = [0,500])
ax3.set(xlabel = 'C', ylabel = 'mean squared error', title = 'MLP - 50 neurons', ylim = [0,500])
ax4.set(xlabel = 'C', title = 'MLP - 100 neurons', ylim = [0,500])

#comparison table for preferred methods
model_LR = Ridge(alpha = 0.5)
model_MLP = MLPRegressor(hidden_layer_sizes=(5), alpha=1/200, max_iter=1000)
model_dummy = DummyRegressor(strategy='mean')
time_LR = 0
time_MLP = 0
time_dummy = 0
final_results_df = pd.DataFrame(columns=['review_metric', 'LR_MSE',
                                         'MLP_MSE', 'Dummy_MSE', 'LR_MAE', 'MLP_MAE', 'Dummy_MAE'])
for i in range(1,8):
    kf = KFold(n_splits=5)
    y = feature_df.iloc[:, i]
    temp_MSE_LR = []
    temp_MSE_MLP = []
    temp_MSE_dummy = []
    temp_MAE_LR = []
    temp_MAE_MLP = []
    temp_MAE_dummy = []
    #LR
    tic = time.time()
    for train, test in kf.split(X):
        model_LR.fit(X[train], y[train])
        if i==1:
            coeff_list_1 = sorted(list(zip(model_LR.coef_,
                                           feature_df.iloc[:, 8:].columns.tolist())),
                                  reverse=True, key=lambda x: x[0])
        ypred = model_LR.predict(X[test])
        temp_MSE_LR.append(MSE(y[test], ypred))
        temp_MAE_LR.append(MAE(y[test], ypred))
    mean_MSE_LR = np.array(temp_MSE_LR).mean()
    mean_MAE_LR = np.array(temp_MAE_LR).mean()
    time_LR += time.time() - tic
    #MLP
    tic = time.time()
    for train, test in kf.split(X):
        model_MLP.fit(X[train], y[train])
        ypred = model_MLP.predict(X[test])
        temp_MSE_MLP.append(MSE(y[test], ypred))
        temp_MAE_MLP.append(MAE(y[test], ypred))
    mean_MSE_MLP = np.array(temp_MSE_MLP).mean()
    mean_MAE_MLP = np.array(temp_MAE_MLP).mean()
    time_MLP += time.time() - tic
    #dummy
    tic = time.time()
    for train, test in kf.split(X):
        model_dummy.fit(X[train], y[train])
        ypred = model_dummy.predict(X[test])
        temp_MSE_dummy.append(MSE(y[test], ypred))
        temp_MAE_dummy.append(MAE(y[test], ypred))
    mean_MSE_dummy = np.array(temp_MSE_dummy).mean()
    mean_MAE_dummy = np.array(temp_MAE_dummy).mean()
    time_MLP += time.time() - tic
    new_row = {'review_metric': feature_df.columns[i].split('_')[2],
               'LR_MSE': mean_MSE_LR, 'MLP_MSE': mean_MSE_MLP, 'Dummy_MSE': mean_MSE_dummy,
               'LR_MAE': mean_MAE_LR, 'MLP_MAE': mean_MAE_MLP, 'Dummy_MAE': mean_MAE_dummy}
    final_results_df = final_results_df.append(new_row, ignore_index=True)
final_results_df.loc[7] = final_results_df.sum()
final_results_df.at[7, 'review_metric'] = 'total'
with pd.option_context('display.max_rows', None, 'display.max_columns', None):
    print(final_results_df)
print('Execution times:')
print('LR: ' + str(time_LR) + ' seconds')
print('MLP: ' + str(time_MLP) + ' seconds')
print('Dummy: ' + str(time_dummy) + ' seconds')

coeff_list_7 = sorted(list(zip(model_LR.coef_,
                               feature_df.iloc[:, 8:].columns.tolist())),
                      reverse=True, key=lambda x: x[0])
print('\n First Metric:')
print(*coeff_list_1, sep='\n')
print('\n Last Metric:')
print(*coeff_list_7, sep='\n')

plt.show()
```