Final Assessment

Data preparation

The AirBnB datasets were downloaded and reviewed. The listings that did not have scores were dropped as they cannot be used for training or testing. The review scores were written to a separate file for easy access. The columns relating to the website that scraped this data from AirBnB eg. scrape_id were ignored as they are irrelevant to listings scores. The columns containing URLs were also ignored as they are not easily turned into features and do not contribute any additional information about the listing. Some fields were simplified, for example, many property type values included “private” or “shared”, but this information was already included in a separate field, so these strings were removed from those values.

The data that was close to being numeric was converted to numeric data. This included converting dates to epoch time, converting boolean values to zeros and ones, stripping percentage signs, etc. This data was combined with the data that was already numeric. The categorical data from the listings file was one hot encoded using the OneHotEncoder from sklearn.preprocessing.

Subset of processed numerical data

id	price	bathrooms	bathrooms shared boolean	last review	host acceptance rate	host is superhost	instant bookable	email verification	work email verification	phone verification
44077	70	1.5	1	1662681600	99	1	0	1	0	1
85156	67	1.5	1	1662854400	99	1	0	1	0	1
159889	45	2	1	1661990400	95	0	0	1	0	1
162809	80	1.5	1	1661904000	87	1	0	1	0	1

Subset of processed categorical data

dun laoghaire rathdown	dublin city	entire home/apt	hotel room	private room
1	0	0	0	1
1	0	0	0	1
0	1	0	0	1
1	0	1	0	0

The textual data (e.g. description) from the listings file was then tokenized. Cross validation using Lasso Regression was run to compare the effect of using different tokenizers, count minimums (min_df), and ngram sizes. The tokenizer generated by CountVectorizer().build_tokenizer() from sklearn.feature_extraction.text performed better than WhitespaceTokenizer, word_tokenize, wordpunct_tokenize, sent_tokenize and RegexpTokenizer from nltk.tokenize. The combination of a count minimum of 90, a minimum ngram size of 1 and a maximum ngram size of 5 yielded the lowest mean squared error (MSE). CountVectorizer was used using nltk.corpus.stopwords along with the aforementioned parameters.

The reviews required more sanitisation. They were all translated to English using the googletrans API, which includes automatic language detection. This took many hours. I decided to take this step so that the same word in different languages was not tokenized as separate features. Reviews for each listing were merged together so that they could be paired with their corresponding row in the listing data.

The BeautifulSoup package was used to remove html syntax from the reviews. Finally, I got the corresponding host name from the listings data and replaced each instance of it in the reviews with “hostname”, as I felt that it would generate more significant tokens than each individual name would. The sanitised reviews were then tokenized using the same parameters and functions as was used for the listings textual data.

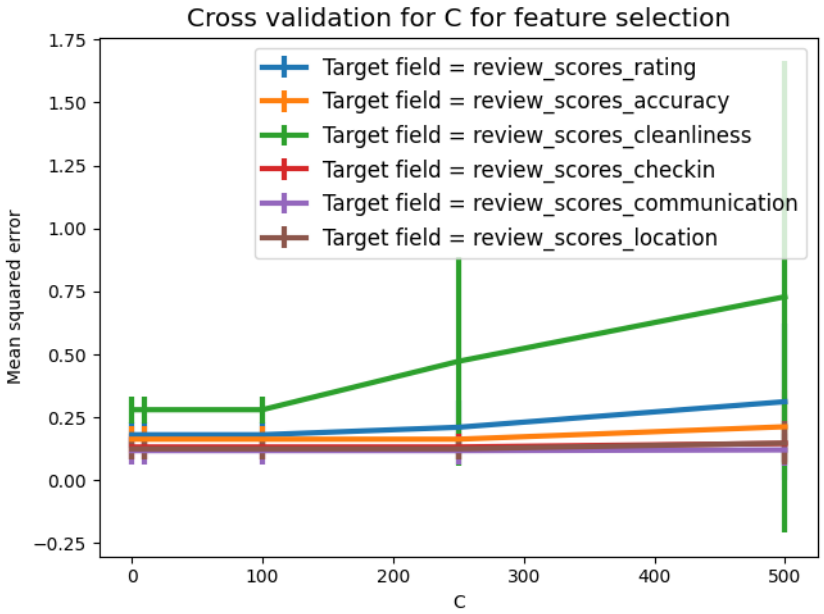
The only dataset with data of different orders of magnitude was the numerical data, therefore, it was normalised using sklearn.preprocessing.normalize.

Feature selection

The useful features were then selected from the prepared data using Lasso Regression. Lasso Regression was selected for this task because it uses the L1 penalty which shrinks the weights of all non-useful features to zero.
The datasets were concatenated and KFold cross-validation for C was executed. k=5 was chosen as it is widely considered that this should be the default as it allows for sufficient training and test data. When C had been selected the models were trained and the weights were collected. Then the features that had a none-zero weight were written into a new file. This process was looped to be performed with each review score.
However, for some review scores, all of the weights had been penalised to zero, and for others, only three or four hadn't been. This meant that the model was concluding that the best approach was to output the same constant every time, with some minor adjustments if certain features were present. I decided to repeat the process on each dataset separately, not on the concatenated datasets together. The cross-validation was performed on every dataset with every review score and the following C values were selected:

Datasets|Score	Rating	Accuracy	Cleanliness	Check in	Communication	Location
Concatenated	100	100	100	0.1	0.1	0.1
Numerical	0.1	0.1	0.1	0.1	0.1	0.1
Categorical	100	0.1	100	0.1	0.1	0.1
Listings text	100	0.1	100	0.1	0.1	100
Reviews text	100	100	100	100	100	100

The values in the above table were selected from plots such as the one opposite. The plot opposite displays the cross validation for C for the numerical data dataset. 0.1 was chosen for each review score as the value that balances the desire for the smallest C value with the desire for the minimum MSE and standard deviation of MSE should be selected.
Processing each dataset separately resulted in more features with non-zero weights. The features with non-zero weights were written to a new file for each review score. These act as the final datasets to be used for training and evaluating the models.



For example the following features were selected (i.e. had non-zero weights) for the "accuracy" review score:

Numerical
None. (This means that features like "host_is_superhost" and "price" are not particularly relevant.)

Categorical
"Neighbourhood cleansed" is "dublin city"
"Property type" is "home" or "rental unit"

Listings text
"Name" includes "centre" or "home"
"Description" includes "10", "15", "access", "apartment", "area", "bed", "bedroom", "bright", "bus", "close", "coffee", "cosy", "double bed", "garden", "guests", "holiday", "home", "house",

Model selection

Ridge Regression

I decided to start with a logistic regression (LR) model as it is good practice to attempt a simple model first before attempting more complicated ones. Logistic regression models assigns a weight to each of the features and sums them: $\Theta^T c = \Theta_0 x_0 + \Theta_1 x_1 + \Theta_2 x_2 + \ldots + \Theta_3 x$. The weights are calculated by minimising the cost function. Different variations of LR use slightly different cost functions. For this problem I did not choose Lasso Regression, as I did not want to further reduce the number of features used, as I had already thinned the data during feature selection and wanted to avoid underfitting. Therefore this time I chose Ridge Regression, which uses a cost function that uses the L2 penalty, as follows $J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\theta^T\theta}{C}$.

This penalty reduces overfitting by reducing the magnitude of the weights.

Neural net

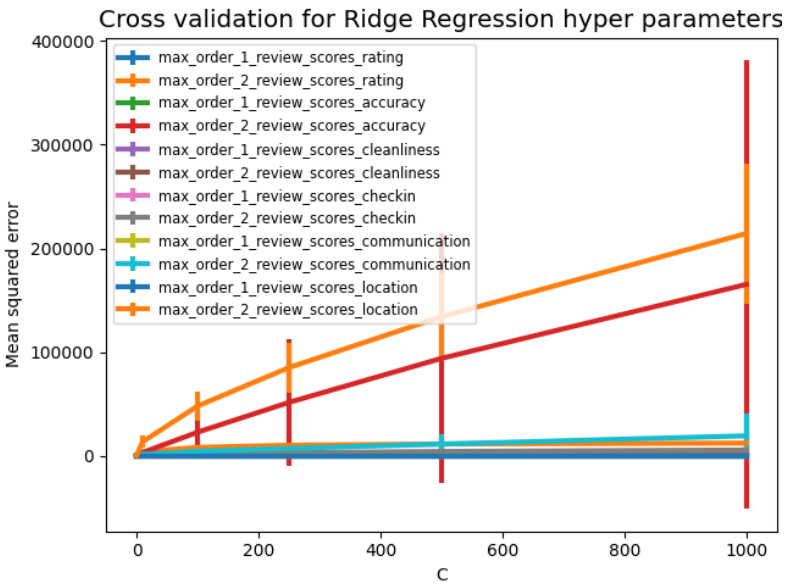
Neural nets are known to scale well relative to other machine learning models, and with the large dataset provided for this project, I believed it could be more performant than others. I also wanted to get some hands-on experience with a deep learning model due to their prevalence in both the tech industry and academia at the moment. I originally intended to apply transfer learning and use a model that had been pre-trained for natural language processing (NLP). I did some research and selected BERT by Facebook because it was touted as the go-to open-source NLP model for transfer learning. (Safarov). However, when I started reading into it further, I realised that to understand it I needed a deep understanding of advanced machine learning concepts and decided not to use transfer learning to avoid the risk of using it without truly understanding it. Instead, I built my own simple neural network.

Method

For all models, I used MSE as the metric for comparison as it is the industry standard for regression models. I also used a train test split of 80:20 as this allows for sufficient data for each purpose.

Ridge Regression

The datasets generated during feature selection were read in. Again, 5-fold cross validation was performed to select the max_order of polynomial features and C, the hyper parameter for the L2 penalty. Applying polynomial features resulted in increased error. Multiple plots were generated with different ranges of C values. The selected values of C were for the review scores are displayed in the table below.

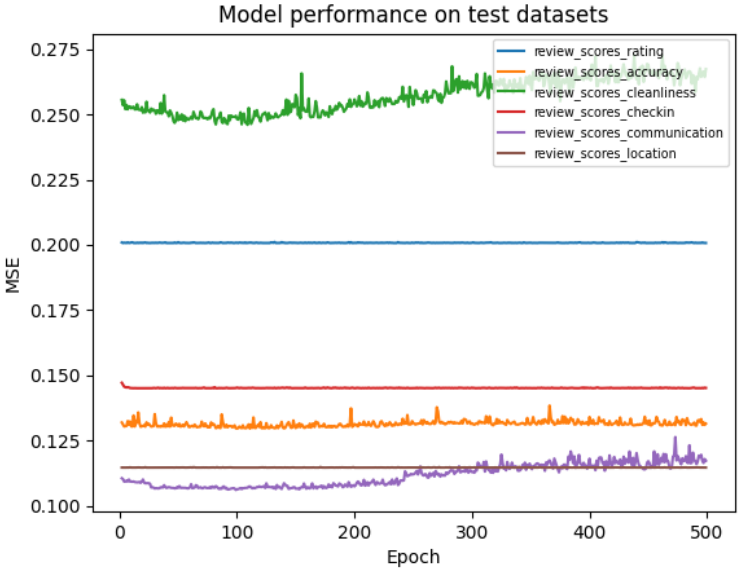


	rating	accuracy	cleanliness	checkin	communication	location
C	0.01	0.001	0.001	0.01	0.01	0.001

The models with the selected values of C were fitted to the training data and evaluated using the test data.

Neural net

A simple neural net was built using Keras. Many iterations were run with different numbers of layers with different output channel sizes and activation functions. A shallow net with small output channel sizes and the ReLU activation function yielded the lowest error. I also added one dropout layer to avoid overfitting.



The neural net was as follows:

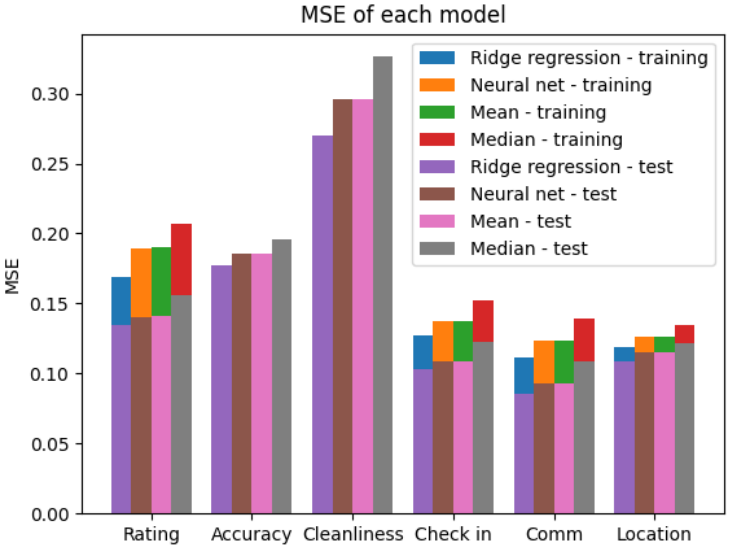
```
Dense(8, activation='relu',
input_shape=Xtrain.shape[1:])
Dense(4, activation='relu')
Dropout(0.5)
Dense(2, activation='relu')
Dense(1)
```

Stochastic Gradient Descent was chosen as the optimizer as it is very scalable and can escape local minima. (Winkler)

The model was trained with the dataset for each score and the history generated during training was plotted to identify the ideal number of epochs that each should be trained with.100 epochs was well suited to all of the review scores as shown in the plot below.
Each model was trained on 100 epochs and the resultant MSE was calculated.

Results

I used the sklearn DummyRegressor function to generate two baseline models - one which always predicts the mean and one that always predicts the median. The models were fit to each dataset and the MSE for each was calculated. The opposite plot compares each model for each review score. It is clear that both the Ridge Regression and Neural Net models perform better than the baselines, but not by a vast amount. Also interestingly, the Ridge Regression model performed marginally better than the Neural Net. The model performed better on the training data than the test data for the rating, check in, communication and location review scores, this indicates some overfitting, despite the dropout layer. The performance of my models follows the same pattern as the mean and median baselines. This implies that the more the data deviates from the mean, the less likely it is that my models are giving an accurate prediction. This is not surprising as the vast majority of the score values are clustered between 4 and 5.



Conclusion

The Ridge Regression model performed the best, proving that it is worthwhile starting with a simple model. However, neither of my models are particularly powerful as they only perform slightly better than the baseline. Upon reflection, perhaps I could have improved my score if I had repeated the cross validation process for the selection of vectorizer parameters for review text, rather than just using the same ones as the listings text. I also may have been able to produce a better score if I had used polynomial features at the feature selection stage, as the combination of features may have yielded insights.
Despite these oversights, my models both beat the baselines and are therefore of value.

Works Cited

Safarov, Safar. “3 Pre-Trained Model Series to Use for NLP with Transfer Learning.” *Towards Data Science*, 5 December 2020, https://towardsdatascience.com/3-pre-trained-model-series-to-use-for-nlp-with-transfer-learning-b2e45c1c275b. Accessed 4 January 2023.

Winkler, Markus. “7 tips to choose the best optimizer | by Davide Giordano.” *Towards Data Science*, 25 July 2020, https://towardsdatascience.com/7-tips-to-choose-the-best-optimizer-47bb9c1219e. Accessed 4 January 2023.

Exam Questions

Give two examples of situations when logistic regression would give inaccurate predictions. Explain your reasoning.

If there are fewer data points than features, logistic regression should not be used as it will lead to overfitting which will mean that when data outside of the training set, the predictions will be inaccurate.

If the data is not linearly separable, logistic regression will not be able to identify patterns in the data and will therefore not be able to make accurate predictions.

Discuss some advantages and disadvantages of a kNN classifier vs an MLP neural net classifier. Explain your reasoning.

kNN models do not require training time, the only calculations that need to be done is when a prediction is being made the distance to the nearby data points must be calculated to identify the nearest neighbours. Therefore, data can be added and will be reflected immediately. Contrary to this MLP neural nets can be very time consuming to train, and when new data becomes available this time consuming process has to be repeated.

On the other hand the kNN does not extrapolate well outside the training set range. If there are no data points near the input data point, it has little to work off and performs poorly. MLP neural networks can learn complex patterns which can make it capable of performing better outside the range of data it was trained with.

kNN is also very sensitive to noisy data. Noisy data points can have a big impact on predictions on input data near it, whereas NLP neural nets learn the patterns and therefore noise that does not fit the pattern is given less weight.

In k-fold cross-validation a dataset is resampled multiple times. What is the idea behind this resampling i.e. why does resampling allow us to evaluate the generalisation performance of a machine learning model.

When a model is trained on a subset of a dataset, the model learns the characteristics of that subset. By using multiple different subsets, the model learns the characteristics of multiple different subsets. When the results of this are combined, any characteristics that are specific to only one or some subset/s are averaged out. In this way random anomalies are ignored and the performance of the model is generalised.

Why are k = 5 or k = 10 often suggested as good choices?

When selecting a number of folds you need to balance the desire to have enough training data against the desire to have enough test data for each iteration of the cross validation. The more folds, the smaller the proportion of the dataset that is used for testing and vice versa. 5 and 10 are considered to be a good point for striking this balance.

Training time is also considered, the more folds, the longer it will take, and each iteration involves training a model which can be computationally intensive. 5 and 10 are not considered to big to create a problem with training time.

Code

preprocessing_of_numerical_values.py

```
import pandas as pd
import datetime
import calendar
import re


def get_epoch_from_date(x):
    if x != 0:
        parts_of_dates = str(x).split('-')
        t = datetime.datetime(int(parts_of_dates[0]), int(parts_of_dates[1]), int(parts_of_dates[2]))
        return calendar.timegm(t.timetuple())
    else:
        return 0


def get_int_from_response_time(x):
    y = 0
    if x == "within an hour":
        y = 60
    elif x == "within a few hours":
        y = 180
    elif x == "within a day":
        y = 1400
    elif x == "a few days or more":
        y = 4320
    return y


def get_int_from_boolean(x):
    if x == 't':
        return 1
    return 0


def get_number_from_bathroom_text(x):
    if x != 0:
        number = re.findall(r'\d*.\d+|\d+', x)
        if len(number) == 1:
            return number[0]
        return 1
    return 0


def get_word_presence_boolean_from_text(x, word):
    if str(x) in str(x).lower():
        return 1
    return 0
```

```
def get_word_presence_boolean_from_list(x, word):
    if word in x:
        return 1
    return 0


def get_int_from_percentage(x):
    if x == '0':
        return x
    return x[:-1]


def map_nearly_numbers():
    listings = pd.read_csv("clean_listings.csv")
    already_numeric = ['id', 'host_id', 'host_listings_count', 'host_total_listings_count', 'latitude',
'longitude',
                       'accommodates', 'bedrooms', 'beds', 'minimum_nights', 'maximum_nights',
'minimum_minimum_nights',
                       'maximum_minimum_nights', 'minimum_maximum_nights',
'maximum_maximum_nights',
                       'minimum_nights_avg_ntm', 'maximum_nights_avg_ntm', 'availability_30',
'availability_60',
                       'availability_90', 'availability_365', 'number_of_reviews', 'number_of_reviews_ltm',
                       'number_of_reviews_l30d', 'calculated_host_listings_count',
                       'calculated_host_listings_count_private_rooms',
'calculated_host_listings_count_shared_rooms',
                       'reviews_per_month']
    numerical = listings[already_numeric]

    numerical["host_location"] = listings["host_location"].apply(
        lambda x: get_word_presence_boolean_from_text(x, "Ireland")
    )
    numerical["host_response_time"] = listings["host_response_time"].apply(
        lambda x: get_int_from_response_time(x)
    )
    numerical["price"] = listings["price"].apply(
        lambda x: str(x)[1:].replace(',', '')
    )
    numerical["bathrooms"] = listings["bathrooms_text"].apply(
        lambda x: get_number_from_bathroom_text(x)
    )
    numerical["bathrooms_text"] = listings["bathrooms_text"].apply(
        lambda x: get_word_presence_boolean_from_text(x, "shared")
    )

    date_values = ["host_since", "first_review", "last_review"]
    for column in date_values:
        numerical[column] = listings[column].apply(lambda x: get_epoch_from_date(x))

    percentage_values = ["host_response_rate", "host_acceptance_rate"]
    for column in percentage_values:
```

```
        numerical[column] = listings[column].apply(lambda x: get_int_from_percentage(x))

    boolean_values = ["host_is_superhost", "host_has_profile_pic", "host_identity_verified",
                      "has_availability", "instant_bookable"]
    for column in boolean_values:
        numerical[column] = listings[column].apply(lambda x: get_int_from_boolean(x))

    verifications = ["email", "work_email", "phone"]
    for method in verifications:
        numerical[method + "_verification"] = listings["host_verifications"].apply(
            lambda x: get_word_presence_boolean_from_list(x, method))

    numerical.to_csv("feature_engineering/numerical_data.csv", index=False)
```

preprocessing_of_class_values.py

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder


def simplify_property_type(x):
    if "Entire " in str(x):
        x = x.replace("Entire ", "")
    if "Private " in str(x):
        x = x.replace("Private ", "")
    if "Shared " in str(x):
        x = x.replace("Shared ", "")
    if "Room in " in str(x):
        x = x.replace("Room in ", "")
    if "room in " in str(x):
        x = x.replace("room in ", "")
    return x


def one_hot_encode():
    df = pd.read_csv("clean_listings.csv")
    df['property_type'] = df['property_type'].apply(lambda x: simplify_property_type(x))
    classification_fields = ["neighbourhood_cleansed", "property_type", "room_type"]
    ohe_df = pd.DataFrame()
    for field in classification_fields:
        encoder = OneHotEncoder()
        df[field] = df[field].apply(lambda x: x.lower())
        ohe = pd.DataFrame(data=encoder.fit_transform(df[[field]]).toarray(),
columns=encoder.categories_)
        ohe_df = pd.concat([ohe_df, ohe], axis=1)

    ohe_df.to_csv("feature_engineering/ohe_listings.csv", index=False)
```

preprocessing_of_textual_values.py

```
import pandas as pd
import numpy as np
from nltk import download
from sklearn.feature_extraction.text import CountVectorizer
from nltk.tokenize import WhitespaceTokenizer
from nltk.tokenize import word_tokenize
from nltk.tokenize import wordpunct_tokenize
from nltk.tokenize import sent_tokenize
from nltk.data import load
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from sklearn.linear_model import Lasso
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
from googletrans import Translator
from bs4 import BeautifulSoup


def get_features_from_listings_text(tokenizers, min_dfs, ngram_min, ngram_max,
ngrams_range_max):
    df = pd.read_csv("clean_listings.csv")
    textual_fields = ['name', 'description', 'neighborhood_overview', 'host_about']

    while ngram_max <= ngrams_range_max:
        for key in tokenizers.keys():
            for min_df in min_dfs:
                print("     with tokenizer=" + key + " and min_df=" + str(min_df)
                      + " and ngram range=(" + str(ngram_min) + "," + str(ngram_max) + ")")
                vectorizer = CountVectorizer(
                    tokenizer=tokenizers[key],
                    stop_words=stopwords.words('english'),
                    min_df=min_df,
                    ngram_range=(ngram_min, ngram_max)
                )

                features = pd.DataFrame()
                for field in textual_fields:
                    data = vectorizer.fit_transform(df[field]).toarray()
                    columns = vectorizer.get_feature_names_out()
                    field_features = pd.DataFrame(data=data, columns=[field + "_" + column for column in
columns])
                    features = pd.concat([features, field_features], axis=1)

                features.to_csv(
                    "feature_engineering/textual_data/"
                    + key + "_" + str(min_df) + "_"
                    + str(ngram_min) + "-" + str(ngram_max)
                    + ".csv", index=False)
        ngram_max += 1
```

```
def vectorization_parameter_cross_validation(c_range, tokenizers, min_dfs, ngram_min, ngram_max,
ngrams_range_max):
    y_df = pd.read_csv("../scores.csv")
    y = y_df['review_scores_rating']

    while ngram_max <= ngrams_range_max:
        for key in tokenizers.keys():
            for min_df in min_dfs:
                print("     with tokenizer=" + key + " and min_df=" + str(min_df)
                      + " and ngram range=(" + str(ngram_min) + "," + str(ngram_max) + ")")
                X = pd.read_csv(
                    "feature_engineering/textual_data/"
                    + key + "_" + str(min_df) + "_"
                    + str(ngram_min) + "-" + str(ngram_max)
                    + ".csv")
                mean_error = []
                std_error = []

                for C in c_range:
                    model = Lasso(alpha=1/(2*C), max_iter=100000)
                    model.fit(X, y)
                    scores = cross_val_score(model, X, y, cv=5, scoring='neg_mean_squared_error')
                    mean_error.append(np.array(scores).mean())
                    std_error.append(np.array(scores).std())

                plt.errorbar(c_range, mean_error, yerr=std_error, linewidth=3,
                             label="tokenizer={} min_df={} ngram range = ({},{})"
                             .format(key, min_df, ngram_min, ngram_max))
        ngram_max += 1

    plt.rc('font', size=12)
    plt.rcParams['figure.constrained_layout.use'] = True
    plt.title("Cross validation for tokenizer, min_kf and ngram range")
    plt.xlabel("C")
    plt.ylabel("Negative mean squared error")
    plt.legend()
    plt.show()


def translate_reviews(index):
    reviews = pd.read_csv("../original_data/reviews.csv")
    translator = Translator()

    # Split up translations because of timeouts
    english_reviews = []
    file = 542
    while index < len(reviews.index):
        english_reviews.append(translator.translate(str(reviews['comments'][index])).text)
        if index % 200 == 0:        # write at regular intervals because of timeouts
            pd.DataFrame(english_reviews)\
                .to_csv("feature_engineering/translations/" + str(file) + ".csv", index=False)
            english_reviews = []
```

```
            file += 1
        index += 1


def merge_translations(max_file_index):
    english_reviews = pd.DataFrame()
    file_index = 2

    while file_index <= max_file_index:
        english_reviews = pd.concat(
            [english_reviews, pd.read_csv("feature_engineering/translations/" + str(file_index) + ".csv")])
        file_index += 1

    english_reviews.columns = ['reviews']
    english_reviews.to_csv("feature_engineering/english_reviews.csv", index=False)


def clean_review(field):
    soup = BeautifulSoup(field, "lxml")
    field = soup.get_text(separator=" ")
    field = field.replace("nan", "")
    field = field.replace("â€™", "'")
    field = field.replace("\r", " ")
    field = field.replace("\t", " ")
    return field


def replace_hostname(df):
    name = df['host_name']
    print(name)
    review = df['reviews']
    return review


def map_reviews_to_listings():
    english_reviews = pd.read_csv("english_reviews.csv")
    original_reviews = pd.read_csv("../original_data/reviews.csv")
    reviews = pd.concat([original_reviews, english_reviews], axis=1)
    reviews['reviews'] = reviews['reviews'].apply(lambda x: clean_review(str(x)))

    listing_reviews = dict()
    for index, entry in reviews.iterrows():
        key = reviews['listing_id'][index]
        value = str(reviews['reviews'][index])
        if listing_reviews.get(key):
            value = str(listing_reviews.get(key)) + " " + value
        listing_reviews.update({reviews['listing_id'][index]: value})

    merged_reviews = pd.DataFrame()
    merged_reviews['listing_id'] = listing_reviews.keys()
    merged_reviews['reviews'] = merged_reviews['listing_id'].apply(lambda x: listing_reviews.get(x))
```

```
    listings = pd.read_csv("clean_listings.csv")
    merged_reviews = pd.merge(
        listings[['id', 'host_name']], merged_reviews, how='inner', left_on='id', right_on='listing_id')

    for index, entry in merged_reviews.iterrows():
        merged_reviews['reviews'][index] = str(merged_reviews['reviews'][index])\
            .replace(str(merged_reviews['host_name'][index]), "hostname")

    merged_reviews.to_csv("feature_engineering/listing_review_map.csv", index=False)


def get_features_from_reviews_text(tokenizer, min_df, ngram_min, ngram_max):
    df = pd.read_csv("listing_review_map.csv")
    df = df.fillna('0')
    vectorizer = CountVectorizer(
        tokenizer=tokenizer,
        stop_words=stopwords.words('english'),
        min_df=min_df,
        ngram_range=(ngram_min, ngram_max)
    )
    data = vectorizer.fit_transform(df['reviews']).toarray()
    columns = vectorizer.get_feature_names_out()
    features = pd.DataFrame(data=data, columns=["review" + "_" + column for column in columns])
    features.to_csv("feature_engineering/ohe_reviews.csv", index=False)


def text_processing():
    packages = ['punkt', 'wordnet', 'omw-1.4', 'stopwords']
    for package in packages:
        download(package)

    c_range = [0.5, 100, 200, 300]
    tokenizers = {
        "CountVectorizer": CountVectorizer().build_tokenizer(),
        # "Whitespace": WhitespaceTokenizer().tokenize,
        # "Word": word_tokenize,
        # "Wordpunct": wordpunct_tokenize,
        # "Sent": sent_tokenize,
        # "Punkt": load('tokenizers/punkt/english.pickle').tokenize,
        # "Regexp": RegexpTokenizer('\w+|\$[\d\.]+|\S+').tokenize
    }
    min_dfs = [70, 80, 90, 100]
    ngram_min = 1
    ngram_max = 5
    ngrams_range_max = 5

    print("Getting features from textual field in listings.csv")
    get_features_from_listings_text(tokenizers, min_dfs, ngram_min, ngram_max, ngrams_range_max)

    print("Cross validating for vectorization parameters")
    vectorization_parameter_cross_validation(c_range, tokenizers, min_dfs, ngram_min, ngram_max,
ngrams_range_max)
```

```
    print("Translating all reviews to English")
    # Takes 8 hours. May time out and need to be run again from where the previous run stopped
    # translate_reviews(1)

    print("Mapping reviews to listings")
    map_reviews_to_listings()

    print("Getting features from reviews.")
    get_features_from_reviews_text(CountVectorizer().build_tokenizer(), 90, 1, 5)
```

feature_selection.py

```
import pandas as pd
import numpy as np
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt
from sklearn.preprocessing import normalize


def read_in_files():
    pre_normalized_numerical = pd.read_csv("numerical_data.csv")
    numerical = pd.DataFrame(
        data=normalize(X=pre_normalized_numerical, norm='max'),
columns=pre_normalized_numerical.columns)
    categorical = pd.read_csv("ohe_listings.csv")
    listings_text = pd.read_csv("textual_data/CountVectorizer_90_1-5.csv")
    reviews_text = pd.read_csv("ohe_reviews.csv")
    pre_normalized_concatenated = pd.concat([numerical, categorical, listings_text, reviews_text],
axis=1)

    return {
        "numerical": numerical,
        "categorical": categorical,
        "listings_text": listings_text,
        "reviews_text": reviews_text,
        "concatenated": pd.DataFrame(
            data=normalize(X=pre_normalized_concatenated, norm='max'),
columns=pre_normalized_concatenated.columns)
    }


def cross_validate_for_c(dataset_name, X):
    y_df = pd.read_csv("../scores.csv")

    for index, score_type in enumerate(y_df.columns):
        print("     for " + score_type)
        kf = KFold(n_splits=5)
        mean_error = []
```

```
        std_error = []

        c_range = [0.1, 10, 100, 250, 500]
        for C in c_range:
            model = Lasso(alpha=1 / (2 * C), max_iter=100000)
            fold = []

            for train, test in kf.split(X):
                model.fit(X.iloc[train], y_df[score_type].iloc[train])
                predictions = model.predict(X.iloc[test])
                fold.append(mean_squared_error(y_df[score_type].iloc[test], predictions))

            mean_error.append(np.array(fold).mean())
            std_error.append(np.array(fold).std())

        plt.errorbar(c_range, mean_error, yerr=std_error, linewidth=3, label="Target field =
{}".format(score_type))

    plt.rc('font', size=12)
    plt.rcParams['figure.constrained_layout.use'] = True
    plt.title("Cross validation for C for feature selection")
    plt.xlabel("C")
    plt.ylabel("Mean squared error")
    plt.legend()
    # plt.show()
    plt.savefig('crossval_{}.png'.format(dataset_name))


def get_useful_features(dataset_name, X):
    y_df = pd.read_csv("../scores.csv")
    weights_df = pd.DataFrame(columns=y_df.columns, index=X.columns)

    selected_c = {
        "numerical": [0.1, 0.1, 0.1, 0.1, 0.1, 0.1],
        "categorical": [100, 0.1, 100, 0.1, 0.1, 0.1],
        "listings_text": [100, 0.1, 100, 0.1, 0.1, 100],
        "reviews_text": [100, 100, 100, 100, 100, 100],
        "concatenated": [100, 100, 100, 0.1, 0.1, 0.1]
    }
    for index, score_type in enumerate(y_df.columns):
        print("     for " + score_type)
        y = y_df[score_type]
        model = Lasso(alpha=1 / (2 * selected_c.get(dataset_name)[index]))
        model.fit(X, y)
        weights_df[score_type] = model.coef_

    weights_df.to_csv("feature_engineering/weights/{}.csv".format(dataset_name))


def list_useful_fields(score, dataset_name, dataset):
    weights = pd.read_csv("feature_engineering/weights/{}.csv".format(dataset_name))
    selected_features = pd.DataFrame()
```

```
    column_names = []
    indices = weights[weights[score] != 0].index.tolist()
    for index in indices:
        column_names.append(weights.iat[index, 0])

    for column in column_names:
        selected_features[column] = dataset[column]
    selected_features.to_csv("feature_engineering/selected/{}_{}.csv".format(score, dataset_name),
index=False)


def merge_final_datasets(score_type, datatypes):
    all_datatypes = []
    for datatype in datatypes:
        try:
            all_datatypes.append(pd.read_csv("feature_engineering/selected/{}_{}.csv".format(score_type,
datatype)))
        except pd.errors.EmptyDataError:
            print("          {} csv was empty.".format(datatype))
            continue

    pd.concat(all_datatypes, axis=1).to_csv("engineered_data/{}_dataset.csv".format(score_type),
index=False)


def feature_selection(y_fields):
    X = read_in_files()

    for dataset in X:
        print("Cross-validating for C for {} features".format(dataset))
        cross_validate_for_c(dataset, X[dataset])

        # print("Getting useful {} features.".format(dataset))
        # get_useful_features(dataset, X[dataset])
        #
        # print("Writing lists of useful {} features.".format(dataset))
        # for score in y_fields:
        #     print("     for " + score)
        #     list_useful_fields(score, dataset, X[dataset])

    # print("Writing final datasets for each score type.")
    # for score_type in y_fields:
    #     print("     for " + score_type)
    #     merge_final_datasets(score_type, list(X.keys())[0:-1])


y_fields = ["review_scores_rating", "review_scores_accuracy", "review_scores_cleanliness",
"review_scores_checkin",
            "review_scores_communication", "review_scores_location"]
feature_selection(y_fields)
```

models.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import KFold
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from sklearn.dummy import DummyRegressor


def setup(score_type):
    y_df = pd.read_csv("scores.csv")
    X = pd.read_csv("engineered_data/{}_dataset.csv".format(score_type))
    y = y_df[score_type]
    return train_test_split(X, y, test_size=0.2)


def cross_validate_for_c(X, y, score_type):
    # max_range = [1, 2]
    max_range = [1]
    for max_order in max_range:
        pf = PolynomialFeatures(max_order)
        Xpoly = pf.fit_transform(X)
        kf = KFold(n_splits=5)
        mean_error = []
        std_error = []

        # c_range = [0.1, 10, 100, 250, 500, 1000]
        c_range = [0.001, 0.01, 0.02, 0.03, 0.04, 0.05]
        for C in c_range:
            print("              with max order = {} and C = {}.".format(max_order, C))
            model = Ridge(alpha=1 / (2 * C), max_iter=100000)
            fold = []

            for train, test in kf.split(Xpoly):
                model.fit(Xpoly[train], y.iloc[train])
                predictions = model.predict(Xpoly[test])
                fold.append(mean_squared_error(y.iloc[test], predictions))

            mean_error.append(np.array(fold).mean())
            std_error.append(np.array(fold).std())

        plt.errorbar(c_range, mean_error, yerr=std_error, linewidth=3,
                     label="max_order_{}_{}".format(max_order, score_type))

    plt.rc('font', size=12)
```

```
    plt.rcParams['figure.constrained_layout.use'] = True
    # plt.title("Cross validation for C for " + score_type)
    plt.title("Cross validation for Ridge Regression hyper parameters")
    plt.xlabel("C")
    plt.ylabel("Mean squared error")
    plt.legend(loc=1, fontsize='x-small')
    # plt.show()
    plt.savefig('ridge_regression/crossval_all_finetuned.png')
    # plt.savefig('ridge_regression/crossval_{}.png'.format(score_type))
    # plt.clf()


def ridge_regression(c, Xtrain, ytrain, Xtest, ytest):
    model = Ridge(alpha=1 / (2 * c), max_iter=100000)
    model.fit(Xtrain, ytrain)
    return [mean_squared_error(ytrain, model.predict(Xtrain)), mean_squared_error(ytest,
model.predict(Xtest))]


def neural_net(epochs, Xtrain, ytrain, Xtest, ytest):
    model = Sequential()
    model.add(Dense(8, activation='relu', input_shape=Xtrain.shape[1:]))
    model.add(Dense(4, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(2, activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='sgd', loss='mean_squared_error')
    history = model.fit(Xtrain, ytrain, epochs=epochs, validation_data=(Xtest, ytest))
    return model, history


def cross_validate_epochs(epochs_total, history, score):
    epochs = np.arange(2, epochs_total)
    mse_train = history.history['loss']
    mse_test = history.history['val_loss']

    # plt.plot(epochs, mse_train[2:], label="training")
    plt.plot(epochs, mse_test[2:], label=score)
    plt.title("Model performance on test datasets")
    plt.xlabel('Epoch')
    plt.ylabel('MSE')
    plt.legend(fontsize='x-small')
    plt.savefig("neural_net/all.png")
    # plt.clf()


def baseline(strategy, Xtrain, ytrain, Xtest, ytest):
    model = DummyRegressor(strategy=strategy)
    model.fit(Xtrain, ytrain)
    return [mean_squared_error(ytrain, model.predict(Xtrain)), mean_squared_error(ytest,
model.predict(Xtest))]
```

```
def generate_bar_chart(ridge_training, nn_training, mean_training, median_training,
                       ridge_test, nn_test, mean_test, median_test):
    labels = ['Rating', 'Accuracy', 'Cleanliness', 'Check in', 'Comm', 'Location']
    x = np.arange(len(labels))
    width = 0.2
    fig, ax = plt.subplots()
    ax.bar(x - 1.5*width, ridge_training, width, label='Ridge regression - training')
    ax.bar(x - 0.5*width, nn_training, width, label='Neural net - training')
    ax.bar(x + 0.5*width, mean_training, width, label='Mean - training')
    ax.bar(x + 1.5*width, median_training, width, label='Median - training')
    ax.bar(x - 1.5*width, ridge_test, width, label='Ridge regression - test')
    ax.bar(x - 0.5*width, nn_test, width, label='Neural net - test')
    ax.bar(x + 0.5*width, mean_test, width, label='Mean - test')
    ax.bar(x + 1.5*width, median_test, width, label='Median - test')
    ax.set_title('MSE of each model')
    ax.set_xticks(x)
    ax.set_xticklabels(labels)
    ax.set_ylabel('MSE')
    ax.legend()
    plt.savefig("results.png")
    # plt.show()


def run_models(y_fields):
    # epochs = [1000, 1000, 1000, 1000, 1000, 1000]
    epochs = [100, 100, 100, 100, 100, 100]
    # epochs = [10, 10, 10, 10, 10, 10]
    # epochs = [1500, 3500, 500, 500, 2000, 500]

    ridge_train = []
    ridge_test = []
    nn_train = []
    nn_test = []
    mean_train = []
    mean_test = []
    median_train = []
    median_test = []

    for index, score_type in enumerate(y_fields):
        print("     for " + score_type)
        X_train, X_test, y_train, y_test = setup(score_type)

        # print("          cross-validating for c")
        # cross_validate_for_c(X_train, y_train, score_type)

        print("          training and testing ridge regression model")
        selected_c = [0.01, 0.001, 0.001, 0.01, 0.01, 0.001]
        mse = ridge_regression(selected_c[index], X_train, y_train, X_test, y_test)
        ridge_train.append(mse[0])
        ridge_test.append(mse[1])
```

```
        print("          training and testing neural net model")
        model, history = neural_net(epochs[index], X_train, y_train, X_test, y_test)
        # cross_validate_epochs(epochs[index], history, score_type)
        nn_train.append(mean_squared_error(y_train, model.predict(X_train)))
        nn_test.append(mean_squared_error(y_test, model.predict(X_test)))

        print("          running baselines")
        mse = baseline('mean', X_train, y_train, X_test, y_test)
        mean_train.append(mse[0])
        mean_test.append(mse[1])
        mse = baseline('median', X_train, y_train, X_test, y_test)
        median_train.append(mse[0])
        median_test.append(mse[1])

    print("          generating bar chart")
    generate_bar_chart(ridge_train, nn_train, mean_train, median_train,
                       ridge_test, nn_test, mean_test, median_test)
```

main.py

```
import pandas as pd
from feature_engineering.preprocessing_of_numerical_values import map_nearly_numbers
from feature_engineering.preprocessing_of_class_values import one_hot_encode
from feature_engineering.preprocessing_of_textual_values import text_processing
from feature_engineering.feature_selection import feature_selection
from models import run_models

df = pd.read_csv("original_data/listings.csv")
y_fields = ["review_scores_rating", "review_scores_accuracy", "review_scores_cleanliness",
"review_scores_checkin",
            "review_scores_communication", "review_scores_location"]
df = df.dropna(subset=y_fields)
df = df.fillna(0)
df.to_csv("feature_engineering/clean_listings.csv", index=False)
df[y_fields].to_csv("feature_engineering/scores.csv", index=False)

print("MAPPING TEXTUAL DATA TO NUMBERS WHERE POSSIBLE")
map_nearly_numbers()

print("ONE-HOT ENCODING CLASSES")
one_hot_encode()

print("GETTING TEXTUAL FEATURES")
text_processing()

print("RUNNING FEATURE SELECTION")
feature_selection(y_fields)

print("RUNNING MODELS")
run_models(y_fields)
```