DUKE UNIVERSITY

Final Project-Kaggle Competition

Zixin Huang (Username: AlessaHuang)
November 19, 2020

1 DATA AND FEATURES

1.1 DATASET

This study focuses on predicting price level, which ranges from 1 to 4, of Airbnb listings in the Buenos Aires, Argentina. The dataset includes 9681 listings. Each listings contains 25 features, including the listing's ID number, neighbourhood, cancellation policy, and features associated with amenities such as room type, bed type and bathrooms. To ensure the training and test dataset to have similar distributions, we remove some training samples, which include categorical values that is not appeared in the test data. This reduces the number of listings in the training set to 9664. To avoid possible influence of the magnitude of features, we standardize all features by removing the mean and scaling to unit variance[1]. When tuning the parameters, we randomly split the training data into 5 folds to perform cross validation.

1.2 EXPLORATORY ANALYSIS

In order to acquire a general view of the correlation between features and price, we create a correlation heatmap (4.1) on the training data with one-hot encoded categorical features. The heatmap shows no significant correlations between all other features and price of the listings. However, there seems to be some correlation between price and bathrooms, beds, guest included, as well as room type.
The location of listings are usually believed to have influences on the prices of the listings. Although in the heatmap we cannot see significant correlations between neighbourhoods and price, the price special distribution[2](1.1) displayed significant price differences between different neighbourhoods.

[1] We did this using **sklearn.preprocessing.StandardScaler()**, see `https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html`
[2] Map source: `http://insideairbnb.com/get-the-data.html`

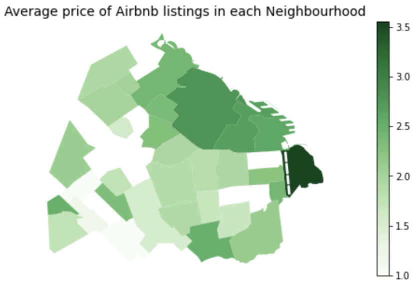


Figure 1.1: Price Special Distribution

Other categorical features are bed type, room type and cancellation policy. From the graphs below(1.2), we can see that most of beds are real beds and there is no much price difference between real bed and pull-on sofa. However, listings with futon or airbed appear to have lower prices. Most of the listings include the entire home or apartment, fewer with only private room, shared room, or hotel room. The price seems to be different for different type of room. We have approximately the same number of listings that apply flexible, moderate, and relatively strict[3] , whereas very few listings have super strict cancellation policy. The plot shows no significant price difference between the first three level of strictness but higher average prices for listings that have super strict cancellation policy.

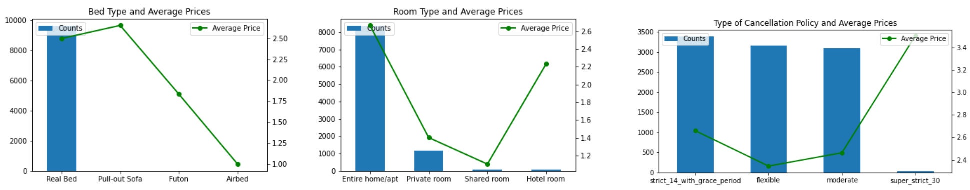


Figure 1.2: Average Price for each type of Bed, Room, and Cancellation Policy

The rest of features are numerical features. Note that for the convenience of analysis, we convert the date for last review and date when host started to number of days since last review and number of days since host to October 8, 2020. And we use 0 and 1 to replace all boolean values.
The histogram(4.2) displays a similar distribution between our training and test dataset. Since all listings in training and test data is not ready for business travel, this feature is useless to our analysis. In addition, from the training data we can see the distribution of the label, which is price level, is balanced.

[3]Strict 14 days with grace period means for a full refund of the nightly rate, the guest must cancel within 48 hours of booking and at least 14 full days prior to listing's local check-in time.
Source: `https://www.airbnb.com/home/cancellation_policies`

2 Models and RESULTS

2.1 Random Forest

Since we cannot find any significant strong relationship between any feature and price level, we do not know whether some features are more important than others. To make use of all the features and investigate their importance, we first apply random forest to train the classification model. This algorithm can be relatively computationally expensive to tune, but it is easy to use and often gives high validation accuracy.

2.1.1 Training

This algorithm fits number of decision trees on sub-samples of the whole training data, then use average results of those trees to produce the final classification. In this case, entropy is used as the criterion for measuring the quality of a tree split. The entropy of a random variable X is defined as

$$H(X) = -\sum_{i=1}^{n} p(x_i)\log_2 p(x_i)$$

where x_i are possible outcomes and $p(x_i)$ are probabilities associated with those outcomes. The decision tree determines which feature to split according to the information gain, which is defined as

$$IG(X,a) = H(X) - H(X|a)$$

where a is a feature. The information gain(IG) of a particular feature can be interpreted as the reduction in the entropy after this feature is known.

2.1.2 Parameters

To prevent overfitting, we first set the number of trees in the forest to be 2000. To find the optimal number of maximum depth of a tree, we use 5-fold cross validation and select the maximum depth that performs the best. The cross validation results are shown in the graph below(2.1).

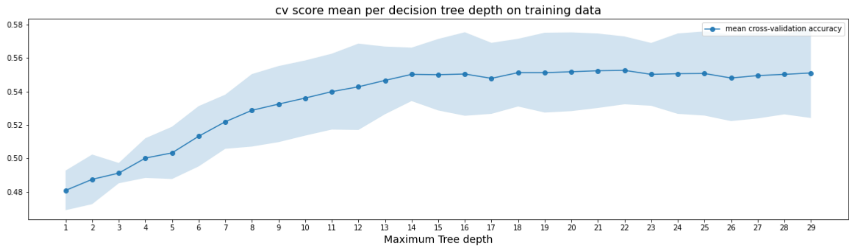


Figure 2.1: Cross Validation Results for Random Forest with All Features

The results shows that the cross validation accuracy increases as the maximum depth increases from 1 to 14, then it converges as the maximum depth increases from 15. The accuracy reaches its highest level, which is around 55.26%, at maximum depth equals to 22.

After running cross validation on different number of trees in the forest, we find that the performance of the algorithm seems not to be affected by this parameter. Therefore, we can keep using 2000 as the number of estimator.

2.1.3 Feature selection

To investigate which features are important to the model, we plot the feature importance using maximum depth equals 15(4.3). As shown in the plot, some features contribute significantly to the model, whereas some other features appear to be not important. For example, the most important features in this model is the cleaning fee of the listings, its feature importance is over 0.12. However, there are many other features, such as whether bed type is airbed, which their importance are less than 0.01. In order to improve efficiency, we drop features which have importance less than 0.01. This gives us the first 21 features in Figure 4.3. Then we do cross validation again on the selected features.
However, selecting only the first 21 important features leads to slighty lower validation accuracy than using all features(4.4). The highest accuracy achieves at maximum depth equals to 22. Run time for cross validation for model with all features is about 3469 seconds, for model with selected features and a smaller range of maximum depth is about 1837 seconds. In this case, feature selection may improve the efficiency but will reduce the accuracy of this algorithm.

2.2 Convolutional Neural Network

Since doing feature selection can reduce accuracy in this case, we want to use all the features in training the second algorithm. To use as much information as possible and find the possible connection between features, we create a convolutional neural network(CNN). This algorithm is fast to train and easy to use, it consists two convolutional layers, four dense layers and dropout regularization. This structure and number of neurons in each layer is similar to those in others work that predicts airbnb prices[4]. We use dropout rate of 25%[5] to prevent overfitting while avoid dropping too much information.

2.2.1 Training

The convolutional layers take the input as a tensor, convolve it according to some fliter and kernel size, then pass the result to the next layer. The dense layers are fully connected layers with element-wise activation function. In this case, we use AdaMax as the optimizer to learn the parameters. Adam is a algorithm for stochastic optimization that estimates the first and second moment of gradients, and update parameters as well as individual learning rates based on the estimates. AdaMax is a variant of Adam based on the infinity norm. Since there are 4 price levels, we use the categorical Cross-Entropy as the loss function. It is a Softmax activation

[4] `https://github.com/L-Lewis/Airbnb-neural-network-price-prediction/blob/master/Airbnb-price-prediction.ipynb`

[5] The validation accuracy actually decreases when the dropout rate is set to 50%, and become worse when the rate is 75%.

adding a Cross-Entropy loss. The formula for Softmax is

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

where $\sigma : \mathbb{R}^K \to \mathbb{R}^K$ is the Softmax activation function, z is the input vector, and K is the number of classes. The formula for categorical Cross-Entropy can be written as

$$CE = -\sum_{i=1}^{K} y_{z,i} \log(\sigma(z)_i)$$

where K represents the number of classes, $y_{z,i}$ equals to 1 if the observation z is in class i and equals to 0 if otherwise, and σ refers to the Softmax activation function.

2.2.2 Parameters

First we tune the learning rate of the algorithm. Since 0.01 is a typical rate for standard multi-layer neural networks[6], we try values around this number. We tentatively use ReLu as activation function. The validation accuracy for each learning rate is shown in Figure 2.2. The graph shows that different learning rate does not lead to significant difference in the validation accuracy, which is around 50%, except when it equals to 0.1, which associated with a much worse accuracy around 25%. Since using 0.01 and 0.005 seems to yield more stable result, we choose 0.005 in this analysis.

Figure 2.2: Validation Accuracy for Different Learning Rate(Left) and Activation Function(Right) at Each Epoch

For the last dense layer we need 4 neurons for 4 classes in the price level, and we use softmax activation function to compute the probability for each class. To determine which activation function to use on other layers, we train models using 4 commonly used activation function and plot their validation accuracy at each epoch(2.2).

The plot suggests that after 15 epochs the performances of algorithm using different activation function are similar. They all achieve accuracy around 50%. We choose ReLu in this analysis.

[6] Y. Bengio. Practical recommendations for gradient-based training of deep architectures,2012.

To evaluate this model[7], we randomly split the training data so that 80% of the data is used for training and the rest 20% of the data is used for validation. Model summary(4.5) can be seen in the Appendix. The result in Figure 2.3 shows that the validation accuracy converges very quickly to about 48%. The training accuracy lies slightly above the validation accuracy, suggesting a slight overfit. The training time for this model is about 62 seconds.

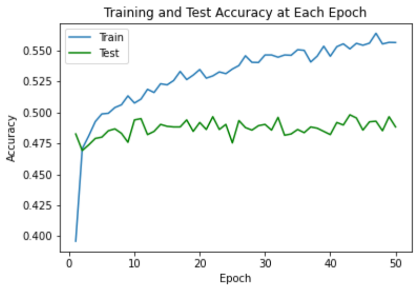


Figure 2.3: Training and Validation Accuracy at each Epoch for the CNN Model

3 Discussion and conclusion

Analysis above shows that, for this dataset, random forest, or perhaps other boosting methods, is more effective in predicting price levels than the deep learning convolution network. Using 20% of the data that is randomly chosen for validation, the validation accuracy for random forest is around 55%(with appropriate maximum depth), whereas for the CNN model is only around 48%. The relatively poor performance of the deep learning model is probably due to the limited number of training sample. However, although random forest gives better accuracy in this case, it is computationally expensive to tune its parameters. Run time for tuning maximum depth for the forest with all features is longer than 3000 seconds, and training the model on the whole training set costs 28 seconds. The CNN model, on the contrary, gives lower accuracy but is faster to tune. In this case, training four CNN models with different activation function and learning rates costs about 600 seconds. Run time for CNN model on the whole training set is 69 seconds.

[7] For batch size, we tried some usual numbers such as 32, 64, 128, and 256 but there seems to be no significant impact on accuracy. Thus, we use 128 in this analysis.

4 Appendix

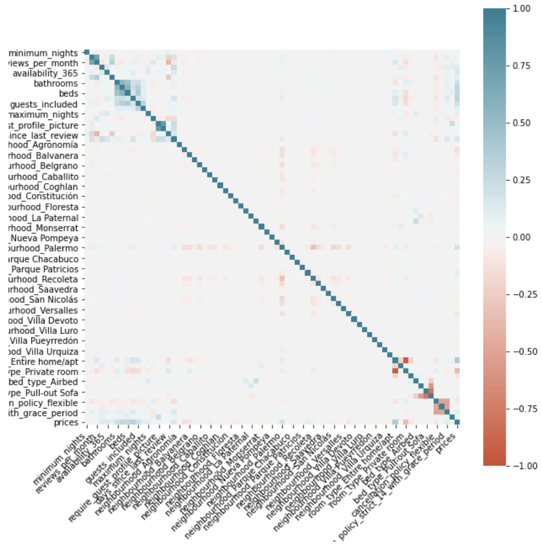


Figure 4.1: Correlation Heatmap of features and price

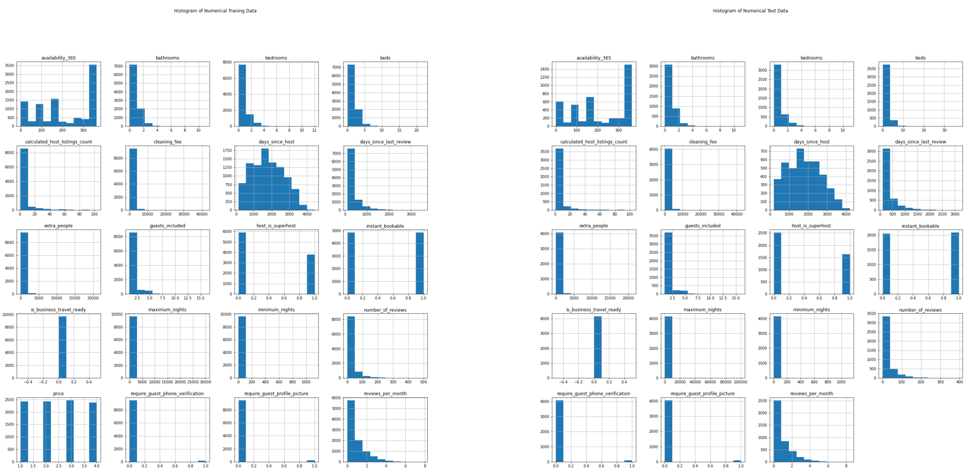


Figure 4.2: Histogram of Numerical Data in the Training(left) and Test(right) Set

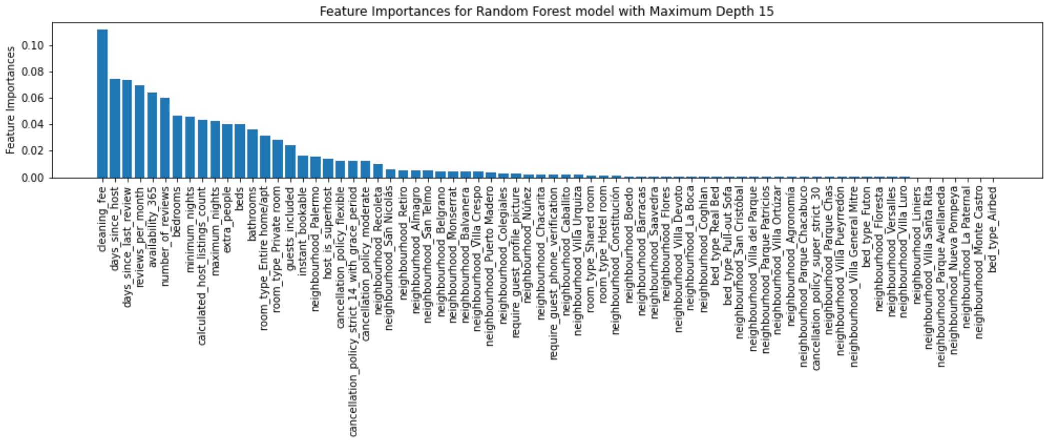


Figure 4.3: Feature Importance of Random Forest

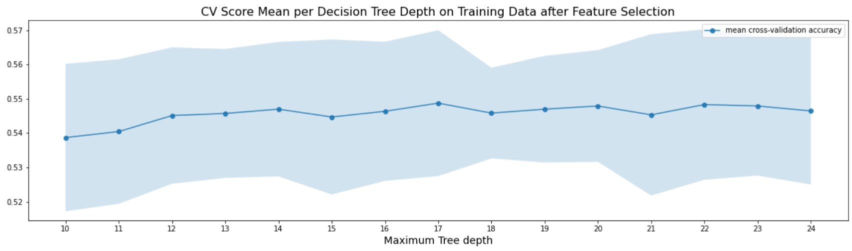


Figure 4.4: Cross Validation Results for Random Forest with Selected Features

```
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 1, 16)             3472
_________________________________________________________________
activation_1 (Activation)    (None, 1, 16)             0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 1, 32)             1568
_________________________________________________________________
activation_2 (Activation)    (None, 1, 32)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 32)             0
_________________________________________________________________
flatten_1 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               4224
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               33024
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               131584
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 2052
_________________________________________________________________
activation_3 (Activation)    (None, 4)                 0
=================================================================
Total params: 175,924
Trainable params: 175,924
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.5: Summary of the CNN model

Final Project

November 19, 2020

1 Final Project

[1]:
```
import pandas as pd
import geopandas as gpd
import numpy as np
import matplotlib.pyplot as plt
import scipy.io as io
import libsvm
from libsvm import svmutil
from svmutil import svm_predict
from libsvm.svmutil import *
%matplotlib inline
import seaborn as sns
import time
from scipy import stats
from datetime import datetime
import sklearn
from sklearn import preprocessing, svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score, KFold
from sklearn.model_selection import train_test_split
```

1.0.1 Training data

[2]:
```
# load training data
data = pd.read_csv('train.csv')
print(data.shape)
```

```
(9681, 25)
```

[3]:
```
# see what features do we have
data.columns
```

[3]:
```
Index(['id', 'neighbourhood', 'room_type', 'minimum_nights',
       'number_of_reviews', 'last_review', 'reviews_per_month',
       'calculated_host_listings_count', 'availability_365', 'host_since',
       'host_is_superhost', 'bathrooms', 'bedrooms', 'beds', 'bed_type',
```

```
        'cleaning_fee', 'guests_included', 'extra_people', 'maximum_nights',
        'instant_bookable', 'is_business_travel_ready', 'cancellation_policy',
        'require_guest_profile_picture', 'require_guest_phone_verification',
        'price'],
       dtype='object')
```

[4]:
```
# drop id
data.drop(columns = 'id', inplace = True)

# change datetime to number of days from 10/8/2020
data['last_review'] = pd.to_datetime(data['last_review'])
data['host_since'] = pd.to_datetime(data['host_since'])
data['days_since_last_review'] = (datetime(2020, 10, 8) - data.last_review).
 ↪astype('timedelta64[D]')
data['days_since_host'] = (datetime(2020, 10, 8) - data.host_since).
 ↪astype('timedelta64[D]')
data.drop(columns=['last_review', 'host_since'], inplace = True)

# replace f/t with 0 and 1
data.replace({'f':0, 't':1}, inplace = True)
```

1.0.2 Test data

[5]:
```
# load test data
data_test =pd.read_csv('test.csv')

# replace f/t with 0 and 1
data_test.replace({'f':0, 't':1}, inplace = True)

# converting date to number
data_test['last_review'] = pd.to_datetime(data_test['last_review'])
data_test['host_since'] = pd.to_datetime(data_test['host_since'])
data_test['days_since_last_review'] = (datetime(2020, 10, 8) - data_test.
 ↪last_review).astype('timedelta64[D]')
data_test['days_since_host'] = (datetime(2020, 10, 8) - data_test.host_since).
 ↪astype('timedelta64[D]')
data_test.drop(columns=['last_review', 'host_since'], inplace = True)

# drop id
ID = data_test.id
data_test.drop(columns = ['id'], inplace = True)
```

1.0.3 Distribution

[6]:
```python
print("Training data:", data.shape)
print('Test data:', data_test.shape)
```

```
Training data: (9681, 24)
Test data: (4149, 23)
```

[7]:
```python
# a quick view of the test data
fig_t = data_test.hist(figsize  = (20,20))
plt.suptitle('Histogram of Numerical Test Data')
plt.savefig('test_his.png')
```

```python
[8]: # a quick view of the training data
fig = data.hist(figsize = (20,20))
plt.suptitle('Histogram of Numerical Traning Data')
plt.savefig('train_his.png')
```

1.0.4 Features

Is_business_travel ready

[9]:
```
# drop is_business_travel_ready
data.drop(columns = ['is_business_travel_ready'], inplace = True)
data_test.drop(columns = ['is_business_travel_ready'], inplace = True)
```

Bed type

[10]:
```
# create table counts each type of bed and the average price of each type in␣
↪training data
k = data['bed_type'].value_counts().to_frame()
k = k.rename(columns = {'bed_type': 'counts'})
a = data.groupby('bed_type').price.mean().to_frame()
k = k.join(a,lsuffix='_caller', rsuffix='_other')
k
```

[10]:

	counts	price
Real Bed	9641	2.496836
Pull-out Sofa	23	2.652174
Futon	12	1.833333
Couch	3	2.000000
Airbed	2	1.000000

[11]:
```
# create table counts each type of bed and the average price of each type in␣
↪test data
k1 = data_test['bed_type'].value_counts().to_frame()
k1.rename(columns = {'bed_type': 'counts'})
```

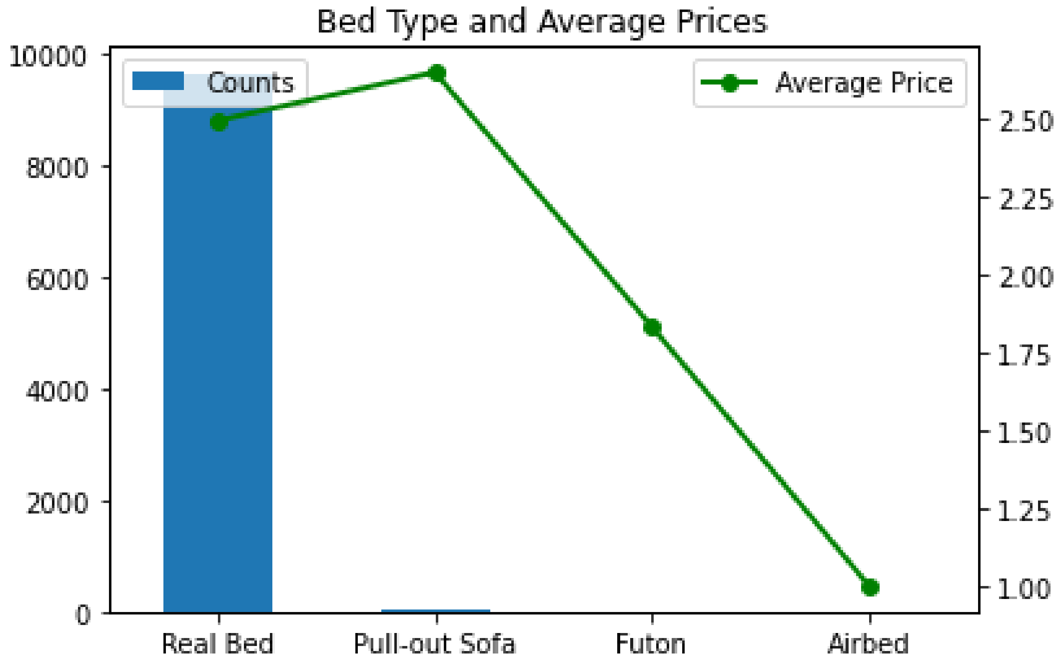
[11]:

	counts
Real Bed	4131
Pull-out Sofa	10
Futon	5
Airbed	3

[12]:
```
# remove listings that have Couch as bed type in the training data
data = data[data.bed_type != 'Couch']
# re-create the table about type and prices
k = data['bed_type'].value_counts().to_frame()
k = k.rename(columns = {'bed_type': 'counts'})
a = data.groupby('bed_type').price.mean().to_frame()
k = k.join(a,lsuffix='_caller', rsuffix='_other')
```

[13]:
```
# plot bed type and average prices for each type in training data
fig = plt.figure()
ax = k['counts'].plot(kind='bar', use_index= True ,label = 'Counts')
ax2 = ax.twinx()
ax2.plot(ax.get_xticks(),
         k['price'],
         linestyle='-',
```

```
         marker='o',color = 'g', linewidth=2.0, label = 'Average Price')
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=0,
)
ax.legend(loc='upper left')
ax2.legend(loc = 'upper right')
ax.set_title('Bed Type and Average Prices')
plt.show()
fig.savefig('bedtype.png')
```



Room type

[14]:
```
# create table counts each type of room and the average price of each type in␣
↪training data
k = data['room_type'].value_counts().to_frame()
k = k.rename(columns = {'room_type': 'counts'})
a = data.groupby('room_type').price.mean().to_frame()
k = k.join(a,lsuffix='_caller', rsuffix='_other')
k
```

[14]:

	counts	price
Entire home/apt	8363	2.663996
Private room	1183	1.402367
Shared room	67	1.104478

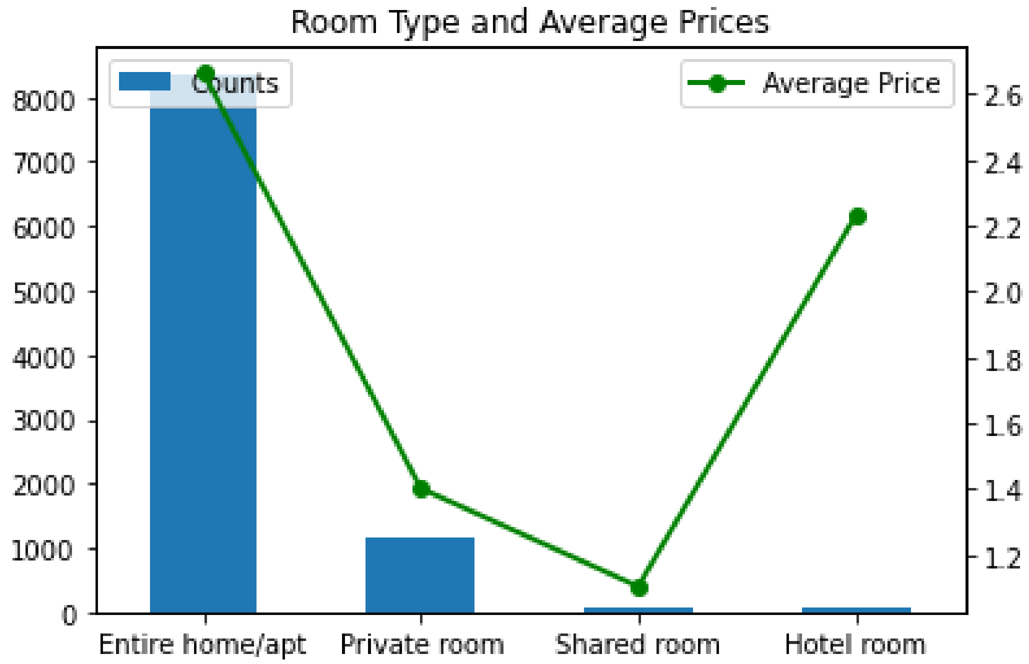
```
Hotel room           65   2.230769
```

[15]:
```
# create table counts each type of room and the average price of each type in␣
 ↪test data
k1 = data_test['room_type'].value_counts().to_frame()
k1.rename(columns = {'room_type': 'counts'})
```

[15]:

	counts
Entire home/apt	3585
Private room	503
Shared room	32
Hotel room	29

[16]:
```
# plot room type and average prices for each type in training data
fig = plt.figure()
ax = k['counts'].plot(kind='bar', use_index= True ,label = 'Counts')
ax2 = ax.twinx()
ax2.plot(ax.get_xticks(),
         k['price'],
         linestyle='-',
         marker='o',color = 'g', linewidth=2.0, label = 'Average Price')
ax.legend(loc='upper left')
ax2.legend(loc = 'upper right')
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=0,
)
ax.set_title('Room Type and Average Prices')
plt.show()
fig.savefig('roomtype.png')
```



Cancellation policy

[17]:
```
k = data['cancellation_policy'].value_counts().to_frame()
k = k.rename(columns = {'cancellation_policy': 'counts'})
a = data.groupby('cancellation_policy').price.mean().to_frame()
k = k.join(a,lsuffix='_caller', rsuffix='_other')
k
```

[17]:

	counts	price
strict_14_with_grace_period	3394	2.659104
flexible	3160	2.344304
moderate	3095	2.462682
super_strict_30	26	3.500000
super_strict_60	3	3.666667

[18]:
```
k1 = data_test['cancellation_policy'].value_counts().to_frame()
k1.rename(columns = {'cancellation_policy': 'counts'})
```

[18]:

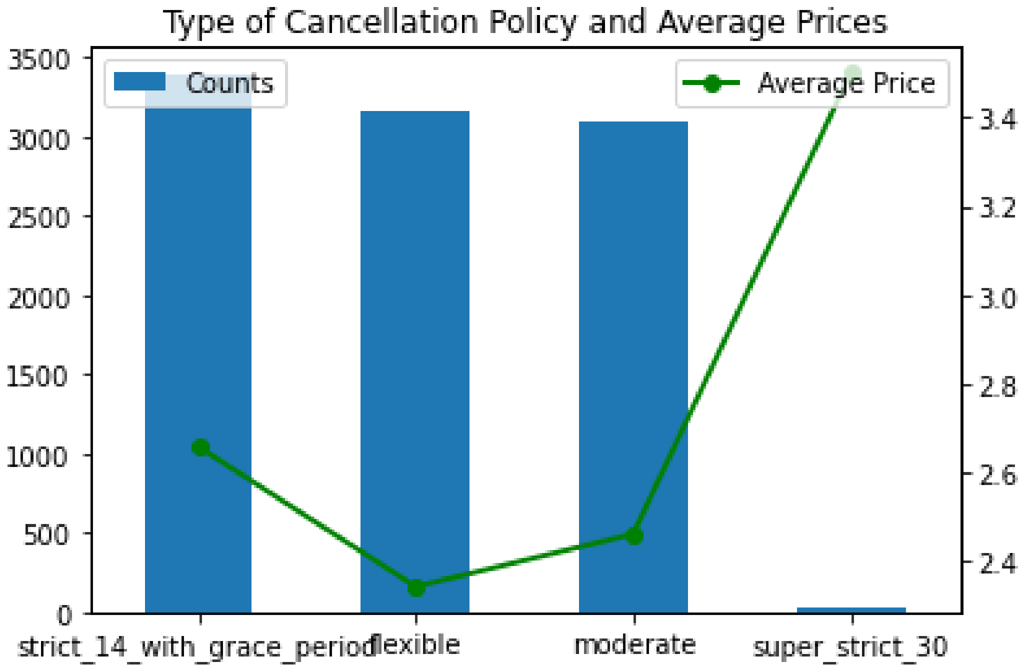
	counts
strict_14_with_grace_period	1464
moderate	1368
flexible	1313
super_strict_30	4

[19]:

```
# remove listings in the training data that has super_strict_60 cancellation␣
 ↪policy
data = data[data.cancellation_policy != 'super_strict_60']

# re-create the table about type and prices
k = data['cancellation_policy'].value_counts().to_frame()
k = k.rename(columns = {'cancellation_policy': 'counts'})
a = data.groupby('cancellation_policy').price.mean().to_frame()
k = k.join(a,lsuffix='_caller', rsuffix='_other')
```

[20]:

```
# plot type of cancellation policy and average prices for each type in the␣
 ↪training data
fig = plt.figure(figsize = (6,4))
ax = k['counts'].plot(kind='bar', use_index= True ,label = 'Counts')
ax2 = ax.twinx()
ax2.plot(ax.get_xticks(),
         k['price'],
         linestyle='-',
         marker='o',color = 'g', linewidth=2.0, label = 'Average Price')
ax.legend(loc='upper left')
ax2.legend(loc = 'upper right')
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=0,
)
ax.set_title('Type of Cancellation Policy and Average Prices')
plt.show()
fig.savefig('cancel.png')
```



Neighbourhood

[21]:
```
k = data['neighbourhood'].value_counts().to_frame()
k = k.rename(columns = {'neighbourhood': 'counts'})
a = data.groupby('neighbourhood').price.mean().to_frame()
k.join(a,lsuffix='_caller', rsuffix='_other')
```

[21]:

	counts	price
Palermo	3299	2.791755
Recoleta	1660	2.685542
San Nicolás	595	2.265546
Retiro	495	2.602020
Belgrano	416	2.413462
Monserrat	390	2.243590
San Telmo	389	2.421594
Almagro	379	1.849604
Balvanera	365	1.939726
Villa Crespo	310	1.951613
Colegiales	182	2.368132
Núñez	175	2.302857
Chacarita	168	2.303571
Caballito	142	1.866197
Puerto Madero	112	3.553571
Villa Urquiza	84	2.000000
Barracas	58	2.120690

```
Constitución            57  1.877193
Saavedra                41  1.926829
La Boca                 39  2.000000
Boedo                   35  1.685714
Flores                  30  1.533333
Coghlan                 28  2.035714
Villa Ortúzar           26  1.500000
Parque Patricios        24  1.666667
Villa Devoto            22  2.090909
Villa del Parque        19  1.947368
San Cristóbal           19  2.052632
Parque Chacabuco        18  1.888889
Parque Chas             17  1.588235
Agronomía               15  2.066667
Villa General Mitre     10  1.200000
Villa Pueyrredón         9  2.000000
Liniers                  8  1.750000
Floresta                 6  2.333333
Vélez Sársfield          6  2.000000
Villa Luro               6  1.166667
La Paternal              4  1.750000
Villa Santa Rita         4  1.750000
Mataderos                3  1.333333
Nueva Pompeya            2  2.500000
Villa Real               2  2.000000
Parque Avellaneda        2  1.000000
Versalles                2  2.500000
Monte Castro             2  1.000000
```

[22]:
```
k = data_test['neighbourhood'].value_counts().to_frame()
k.rename(columns = {'neighbourhood': 'counts'})
```

[22]:

	counts
Palermo	1372
Recoleta	707
San Nicolás	247
Retiro	204
Belgrano	199
Balvanera	178
Monserrat	175
San Telmo	164
Almagro	153
Villa Crespo	148
Colegiales	79
Núñez	77
Chacarita	71
Caballito	64

```
Puerto Madero            39
Villa Urquiza            38
Barracas                 31
Constitución             31
Saavedra                 22
San Cristóbal            18
La Boca                  15
Boedo                    13
Parque Chas              13
Villa Devoto             12
Villa Ortúzar            11
Villa del Parque         10
Flores                    8
Parque Patricios          7
Agronomía                 7
La Paternal               6
Floresta                  6
Villa Pueyrredón          5
Coghlan                   4
Parque Avellaneda         3
Monte Castro              2
Villa Luro                2
Nueva Pompeya             2
Parque Chacabuco          2
Versalles                 1
Villa Santa Rita          1
Liniers                   1
Villa General Mitre       1
```

[23]:

```
# neighbourhoods not included in the test data
cols = []
for i in set(data["neighbourhood"]):
    if i not in set(data_test["neighbourhood"]):
        print(i)
        cols.append(i)
```

```
Vélez Sársfield
Villa Real
Mataderos
```

[24]:

```
# remove listings in the training data that is in neighbourhoods above
data = data[data.neighbourhood != 'Mataderos']
data = data[data.neighbourhood != 'Vélez Sársfield']
data = data[data.neighbourhood != 'Villa Real']
```

[25]:

```
print("Training data:", data.shape)
print('Test data:', data_test.shape)
```

```
Training data: (9664, 23)
Test data: (4149, 22)
```

[26]:
```
# one-hot encoded categorical features
data_test = pd.get_dummies(data_test)
Xdata = pd.get_dummies(data)
print("Training data:", Xdata.shape)
print('Test data:', data_test.shape)
```

```
Training data: (9664, 73)
Test data: (4149, 72)
```

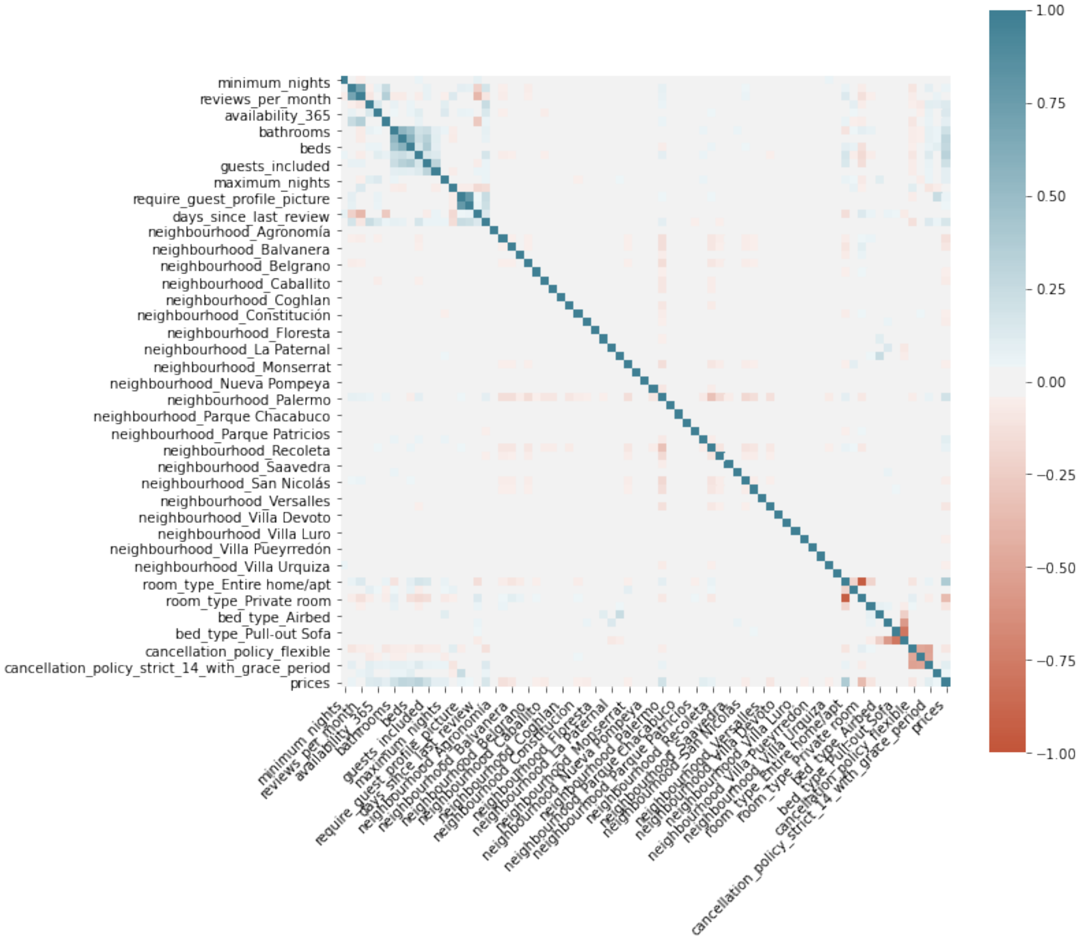
[27]:
```
# seperate price from features
Y = Xdata.price
Xdata.drop(columns = 'price', inplace = True)
print("Training data:", Xdata.shape)
```

```
Training data: (9664, 72)
```

1.1 Exploratory Analysis

[28]:
```
# add prices in the last column for visualization
Xdata['prices'] = Y
```

[29]:
```
# create a correlation heatmap

corr = Xdata.corr()
fig = plt.figure(figsize = (10,10))
ax = sns.heatmap(
    corr,
    vmin=-1, vmax=1, center=0,
    cmap=sns.diverging_palette(20, 220, n=200),
    square=True
)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation=45,
    horizontalalignment='right'
);
plt.show()

fig.savefig('heatmap.png')
```



[30]:
```python
# drop prices from the features
Xdata.drop(columns = 'prices', inplace = True)
```

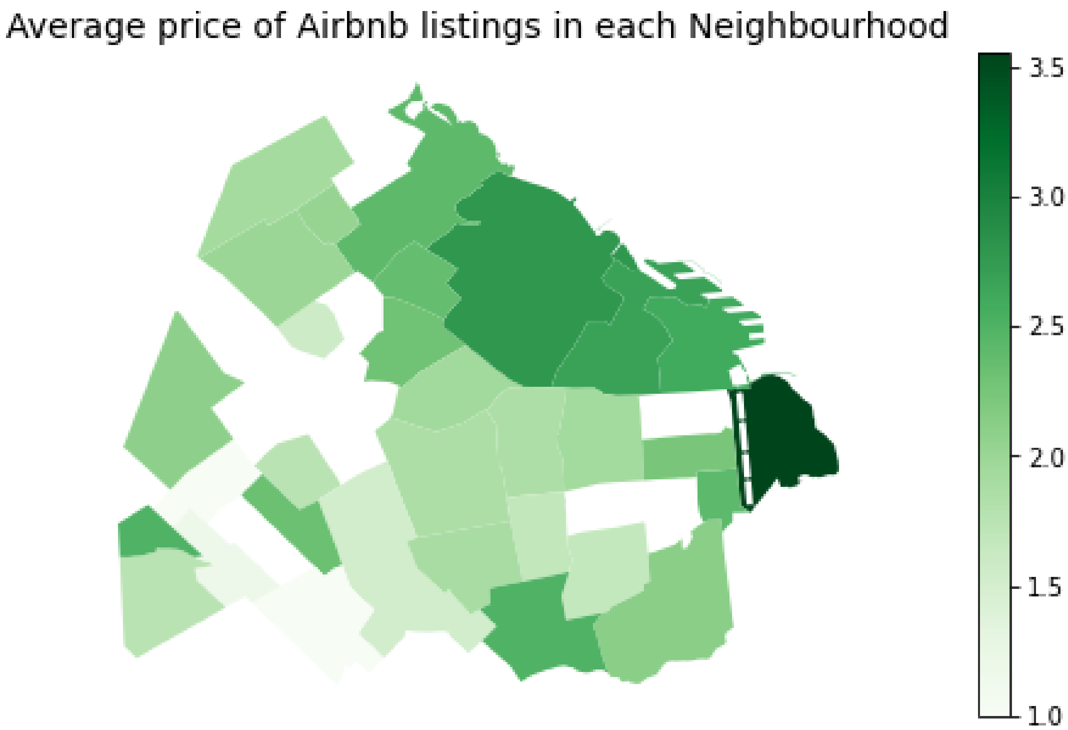
1.1.1 Neighbourhood

[31]:
```python
# Use the geojson file of the Buenos Aires neighbourhood
map_neighbour = gpd.read_file('neighbourhoods.geojson')
map_neighbour.drop(columns = ['neighbourhood_group'], inplace = True)
```

Source: http://insideairbnb.com/get-the-data.html

[32]:
```python
# create a dataframe with average prices and number of listings for each␣
↪neighbourhood
n = pd.DataFrame(data.groupby('neighbourhood').size())
n.rename(columns={0: 'number_of_listings'}, inplace=True)
n['avg_price'] = data.groupby('neighbourhood').price.mean().values
n['median_price'] = data.groupby('neighbourhood').price.median().values
```

```
# combine the dataframe and the geo-map
n_map = map_neighbour.set_index('neighbourhood').join(n)

# Plot the average price of listings in each borough
fig2, ax2 = plt.subplots(1, figsize=(15, 5))
n_map.plot(column='avg_price', cmap='Greens', ax=ax2)
ax2.axis('off')
ax2.set_title('Average price of Airbnb listings in each Neighbourhood',␣
 ↪fontsize=14)
sm = plt.cm.ScalarMappable(cmap='Greens', norm=plt.Normalize(vmin=min(n_map.
 ↪median_price),
                                                            vmax=max(n_map.
 ↪avg_price)))
sm._A = [] # Creates an empty array for the data range
cbar = fig2.colorbar(sm)
plt.show()
fig2.savefig('neigh.png')
```



2 Models

2.0.1 Prepare data for training

[33]:
```
# scale features
scaler = preprocessing.StandardScaler()
# scale training data
X = pd.DataFrame(scaler.fit_transform(Xdata), columns=list(Xdata.columns))
# scale test data
X_test = pd.DataFrame(scaler.fit_transform(data_test), columns=list(data_test.
↪columns))
```

[34]:
```
X.head()
```

[34]:
```
   minimum_nights  number_of_reviews  reviews_per_month  \
0       -0.195658           3.984677           1.532531
1       -0.195658          -0.353632          -0.370085
2       -0.104399          -0.599196          -0.858190
3       -0.150029          -0.626481          -0.808384
4       -0.150029           0.192068           0.596165

   calculated_host_listings_count  availability_365  host_is_superhost  \
0                       -0.145142          0.992381           1.247643
1                       -0.386370         -1.574888           1.247643
2                        1.060994          1.096259          -0.801511
3                       -0.386370          0.406212          -0.801511
4                       -0.064733          1.133358           1.247643

   bathrooms  bedrooms      beds  cleaning_fee  …  room_type_Private room  \
0  -0.438704 -1.383005 -0.663041      0.386720  …               -0.372761
1  -0.438704 -0.164427 -0.663041     -0.532227  …                2.682681
2  -0.438704 -1.383005 -0.663041     -0.236295  …               -0.372761
3  -0.438704 -0.164427  0.802900     -0.866498  …               -0.372761
4  -0.438704 -0.164427  0.802900      0.386720  …               -0.372761

   room_type_Shared room  bed_type_Airbed  bed_type_Futon  \
0              -0.083554        -0.014387        -0.03526
1              -0.083554        -0.014387        -0.03526
2              -0.083554        -0.014387        -0.03526
3              -0.083554        -0.014387        -0.03526
4              -0.083554        -0.014387        -0.03526

   bed_type_Pull-out Sofa  bed_type_Real Bed  cancellation_policy_flexible  \
0               -0.048843           0.061995                     -0.695886
1               -0.048843           0.061995                     -0.695886
2               -0.048843           0.061995                     -0.695886
3               -0.048843           0.061995                      1.437017
4               -0.048843           0.061995                     -0.695886
```

```
   cancellation_policy_moderate  \
0                     -0.685916
1                      1.457904
2                      1.457904
3                     -0.685916
4                      1.457904

   cancellation_policy_strict_14_with_grace_period  \
0                                         1.359491
1                                        -0.735569
2                                        -0.735569
3                                        -0.735569
4                                        -0.735569

   cancellation_policy_super_strict_30
0                            -0.051939
1                            -0.051939
2                            -0.051939
3                            -0.051939
4                            -0.051939

[5 rows x 72 columns]
```

[35]:
```
# define cross validation folds: 5-folds cross validation
kf = KFold(n_splits = 5, shuffle=True, random_state = 15)
```

[36]:
```
# define a function that plots cross validation results and save the plot
def plot_cross_validation_results(depths, cv_scores_mean,
                                   cv_scores_std, title):
    fig, ax = plt.subplots(1,1, figsize=(20,5))
    ax.plot(depths, cv_scores_mean, '-o',
            label='mean cross-validation accuracy', alpha=0.9)
    ax.fill_between(depths, cv_scores_mean-2*cv_scores_std,
                    cv_scores_mean+2*cv_scores_std, alpha=0.2)
    ylim = plt.ylim()
    ax.set_title(title, fontsize=16)
    ax.set_xlabel('Maximum Tree depth', fontsize=14)
    ax.set_xticks(depths)
    ax.legend()
    fig.savefig(title + '.png')
```

2.1 Random Forest

2.1.1 All Features

Tuning max_depth

[54]:
```
# define a function that runs cross validation on trees
def cross_val_trees(x, y, kf, depths):

    """

    This function runs cross validation on random forest with differnt␣
↪max_depth, given cross validation split
    Input: x - data, y - label, kf - k fold cv spliting, depths - range of␣
↪max_depth
    Output: mean cross validation accuracy, cross validation standard␣
↪deviation, run time of this function

    """

    # record run time
    start = time.time()

    cv_scores_mean, cv_scores_std = [], []

    for depth in depths:
        clf = RandomForestClassifier(n_estimators=2000, criterion='entropy',␣
↪max_depth = depth, oob_score = True,
                                    random_state=0,verbose=0)
        scores_clf = cross_val_score(clf, x, y, scoring='accuracy', cv=kf)

        # cv results
        cv_scores_mean.append(scores_clf.mean())
        cv_scores_std.append(scores_clf.std())
        print(depth, scores_clf.mean())


    cv_scores_mean = np.array(cv_scores_mean)
    cv_scores_std = np.array(cv_scores_std)

    run_time = time.time()-start

    return cv_scores_mean, cv_scores_std, run_time
```

[55]:
```
# possible max_depth
depths = range(1,30)
```

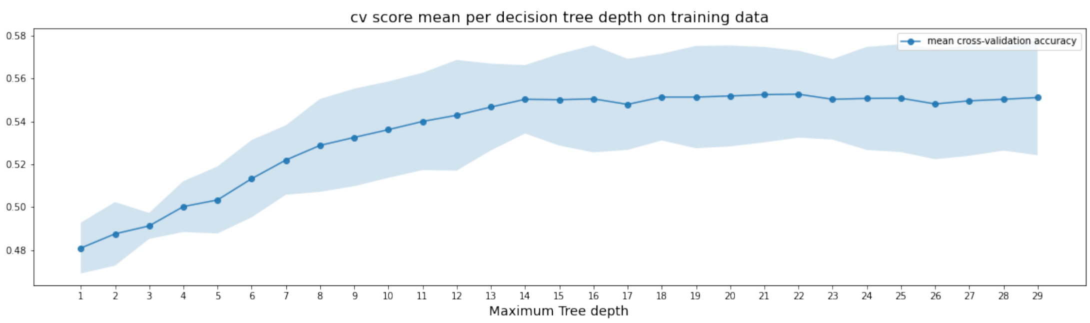
[56]:
```
# run the cross validation function
rdf_mean, rdf_std, rdf_time = cross_val_trees(X, Y, kf, depths)
print(rdf_time)
```

```
1 0.48085775122932956
2 0.48748038588790743
```

```
3 0.4912049518068547
4 0.5002074677685916
5 0.5033122545223583
6 0.5132461797332801
7 0.5219378689193575
8 0.5287679177926371
9 0.5324930192504812
10 0.5361149759168158
11 0.5399437041511762
12 0.5428413444596896
13 0.5466696442629324
14 0.5502911189442601
15 0.550084561591793
16 0.5504989615900792
17 0.5479116125183288
18 0.5513258872005132
19 0.5513264762932997
20 0.5518435390980883
21 0.5524644428949519
22 0.5526715357863157
23 0.5502911724981497
24 0.5507053047269876
25 0.550808770841835
26 0.5481188660713616
27 0.5495668561403283
28 0.5502916544831568
29 0.5511196511713843
3469.4145352840424
```

[57]:
```python
# plot the cv score mean for each max_depth
fig = plt.figure()
plot_cross_validation_results(depths, rdf_mean, rdf_std,
                              'cv score mean per decision tree depth on␣
 ↪training data')
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



```
[77]: # choose the best max_depth and train on the whole trainging set
max_dep = 22
s = time.time()
clf_rdf = RandomForestClassifier(n_estimators=2000, criterion='gini', max_depth␣
 ↪= max_dep, random_state=0)
clf_rdf.fit(X, Y)
print('training acc:', clf_rdf.score(X, Y), 'time:', time.time()-s)
```

```
training acc: 0.9995860927152318 time: 27.356616020202637
```

Tuning n_estimators

```
[82]: # define a function that runs cross validation on trees
def cross_val_trees_nestimator(x, y, kf, n):

    """

    This function runs cross validation on random forest with differnt␣
 ↪n_estimators, given cross validation split
    Input: x - data, y - label, kf - k fold cv spliting, depths - range of␣
 ↪max_depth
    Output: mean cross validation accuracy, cross validation standard␣
 ↪deviation, run time of this function

    """

    # record run time
    start = time.time()

    cv_scores_mean, cv_scores_std = [], []

    for num in n:
        clf = RandomForestClassifier(n_estimators=num, criterion='entropy',␣
 ↪max_depth = 22, oob_score = True,
                                         random_state=0,verbose=0)
        scores_clf = cross_val_score(clf, x, y, scoring='accuracy', cv=kf)

        # cv results
        cv_scores_mean.append(scores_clf.mean())
        cv_scores_std.append(scores_clf.std())
        print(depth, scores_clf.mean())


    cv_scores_mean = np.array(cv_scores_mean)
    cv_scores_std = np.array(cv_scores_std)
```

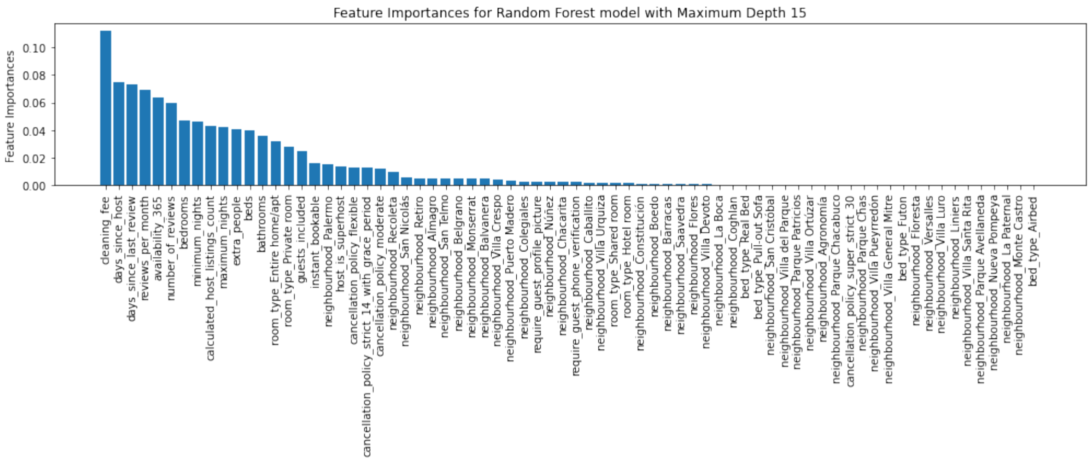
```
    run_time = time.time()-start

    return cv_scores_mean, cv_scores_std, run_time
```

[84]:
```
# possible number of trees in the forest
n = [100, 500, 1000, 1500, 2000]
# run the cross validation function
rdf_n_mean, rdf_n_std, rdf_n_time = cross_val_trees(X, Y, kf, n)
print(rdf_n_time)
```

```
100 0.5493599239106335
500 0.5493599239106335
1000 0.5493599239106335
1500 0.5493599239106335
2000 0.5493599239106335
703.7198657989502
```

2.1.2 Feature importances

[61]:
```
# sort the importances from high to low and  plot importance for each feature
imp=[]
for i,j in zip(X.columns, range(len(clf_rdf.feature_importances_))):
    imp.append((i,clf_rdf.feature_importances_[j]))
imp.sort(key = lambda x: -x[1])

fig = plt.figure(figsize = (14,6))
plt.bar([x[0] for x in imp], np.abs([x[1] for x in imp]))
plt.xticks(rotation=90)
plt.title('Feature Importances for Random Forest model with Maximum Depth 15')
plt.ylabel('Feature Importances')
plt.tight_layout() # make room for xlabels
plt.show()
fig.savefig('feature_importances_rdf.png')
```



[62]:
```
# drop features which importance is less than 0.01
delfe = []
for i in range(len(X.columns)):
    if clf_rd.feature_importances_[i]<0.01:
        delfe.append(X.columns[i])

new_X = X.copy()
new_X.drop(columns = delfe, inplace = True)
```

[63]:
```
new_X.shape
```

[63]: (9664, 22)

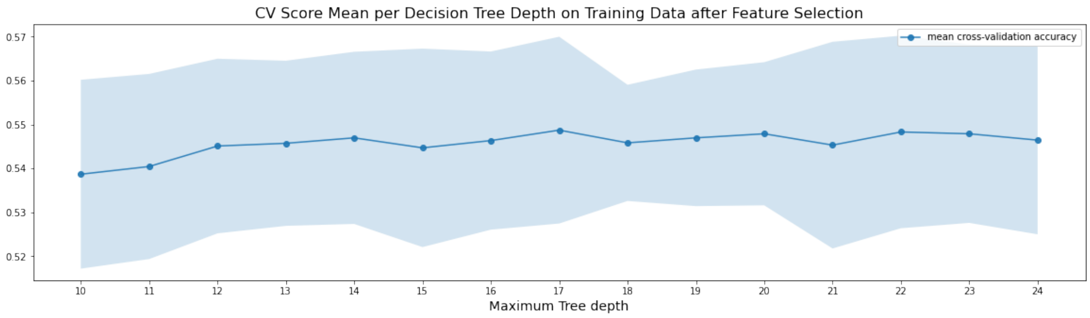
[65]:
```
# run cross validation on the selected features

# possible max_depth
new_depths = range(10,25)

# run the cross validation function
new_rdf_mean, new_rdf_std, new_rdf_time = cross_val_trees(new_X, Y, kf,␣
↪new_depths)
print(new_rdf_time)
```

```
10 0.5387013610185521
11 0.5404607669559647
12 0.5451170634474353
13 0.5457380743520783
14 0.5469799890535849
15 0.5447039487425012
16 0.5463591923645007
17 0.5487392343293286
```

```
18 0.5458412726974773
19 0.546979667730247
20 0.5479108627638734
21 0.5453245312160268
22 0.5483253698699391
23 0.547911076979432
24 0.5464624442637893
1837.7601029872894
```

[66]:
```
# plot the cv score mean for each max_depth
fig = plt.figure()
plot_cross_validation_results(new_depths, new_rdf_mean, new_rdf_std,
                              'CV Score Mean per Decision Tree Depth on␣
 ↪Training Data after Feature Selection')
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



2.2 CNN

[37]:
```
import tensorflow as tf
import keras
from tensorflow.python.keras import backend as K
from keras.models import Sequential
from keras.layers import Conv1D, MaxPooling1D, Flatten, Dense, Activation,␣
 ↪Dropout
from keras.layers.advanced_activations import LeakyReLU
```

```
Using TensorFlow backend.
```

[38]:
```
# define a function that reset tensorflow session
def reset_tf_session():
    """
    A function that clears tf session/graph
```

```
    """
    curr_session = tf.compat.v1.get_default_session()
    # close current session
    if curr_session is not None:
        curr_session.close()
    # reset graph
    K.clear_session()
    # create new session
    config = tf.compat.v1.ConfigProto()
    config.gpu_options.allow_growth = True
    s = tf.compat.v1.InteractiveSession(config=config)
    K.set_session(s)
    return s
```

[39]:
```
# one-hot encode price levels
Y_oh = pd.get_dummies(Y)
print("Train samples:", X.shape, Y_oh.shape)
```

```
Train samples: (9664, 72) (9664, 4)
```

[40]:
```
# split data into training and validation set
X_tr, X_val, Y_tr, Y_val = train_test_split(X, Y_oh, test_size=0.2, shuffle =␣
↪True)
X_tr = X_tr.to_numpy()
X_val = X_val.to_numpy()
Y_tr = Y_tr.to_numpy()
Y_val = Y_val.to_numpy()
# add 1 dim to the array
X_tr=np.expand_dims(X_tr,axis=1)
X_val=np.expand_dims(X_val,axis=1)
```

2.2.1 Learning Rate

[43]:
```
# define possible learning rate
eta = [5e-4, 1e-3, 5e-3, 0.01, 0.05, 0.1]
```

[44]:
```
# first try relu as activation function
def make_cnnmodel():
    """
    Define the model architecture.

    """
    model = Sequential()

    model.add(Conv1D(filters = 16, kernel_size = 3, padding = 'same',␣
↪input_shape=(1,72)))
    model.add(Activation('relu'))
```

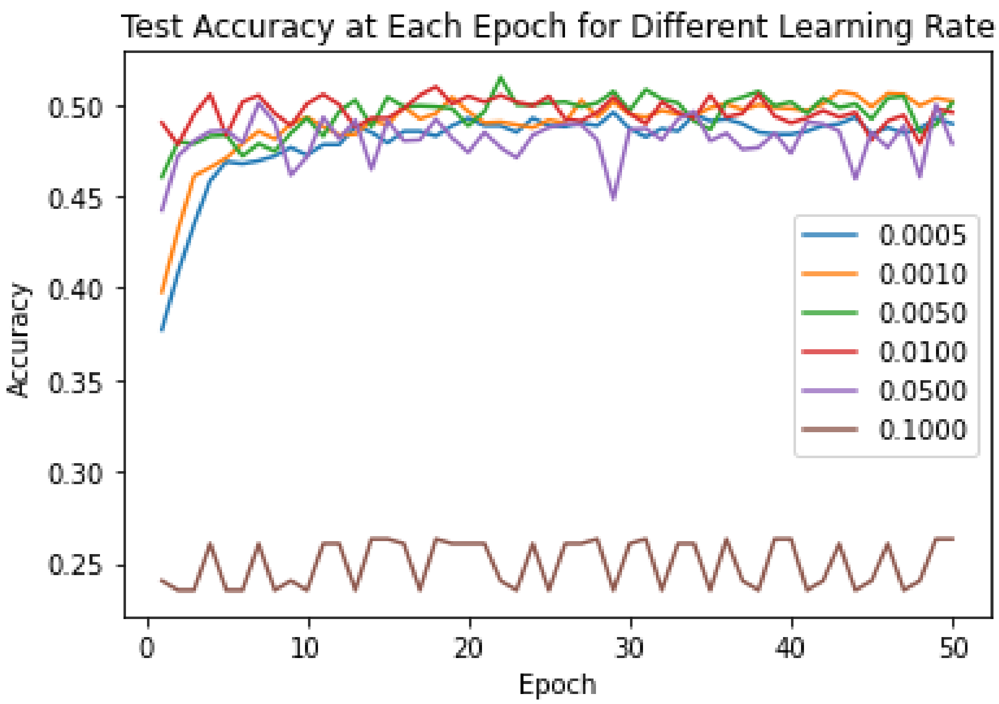
```
    model.add(Conv1D(filters = 32, kernel_size = 3, padding = 'same'))
    model.add(Activation('relu'))
    model.add(Dropout(0.25))


    model.add(Flatten(input_shape=(1,72)))
    model.add(Dense(128, activation = 'relu'))
    model.add(Dropout(0.25))
    model.add(Dense(256, activation = 'relu'))
    model.add(Dropout(0.25))
    model.add(Dense(512, activation = 'relu'))
    model.add(Dropout(0.25))
    model.add(Dense(4))
    model.add(Activation('softmax'))

    return model
```

[57]:
```
# store results
val_acc_lr = []
start = time.time()
for e in eta:
    # initial learning rate
    init_lr = e
    batch = 128
    epoch = 50

    # clear default graph
    s = reset_tf_session()

    # define the model
    model = make_cnnmodel()

    # prepare model for fitting
    model.compile(
        loss='categorical_crossentropy',  # train 4-way classification
        optimizer=keras.optimizers.adamax(lr = init_lr),
        metrics=['accuracy']  # report accuracy during training
    )

    # fit the model
    model.fit(
    X_tr, Y_tr,
    batch_size = batch,
    epochs = epoch,
    validation_data=(X_val, Y_val),
    shuffle=True,
    verbose=0,
```

```python
    initial_epoch=0
    )

    # get validation results
    val = model.history.history['val_accuracy']
    val_acc_lr.append(val)
t = time.time()-start
print('run time:', t)
```

run time: 362.9908571243286

[58]:
```python
# plot the result
fig = plt.figure()
for i in range(len(eta)):
    plt.plot(range(1,epoch+1), val_acc_lr[i], label = '%.4f'%eta[i])

plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Test Accuracy at Each Epoch for Different Learning Rate')
plt.show()
fig.savefig('Learning Rate.png')
```



2.2.2 Activation functions

ReLu

[59]:
```
# describe model
s = reset_tf_session()
model = make_cnnmodel()

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_1 (Conv1D)            (None, 1, 16)             3472
_________________________________________________________________
activation_1 (Activation)    (None, 1, 16)             0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 1, 32)             1568
_________________________________________________________________
activation_2 (Activation)    (None, 1, 32)             0
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 32)             0
_________________________________________________________________
flatten_1 (Flatten)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               4224
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               33024
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               131584
_________________________________________________________________
dropout_4 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 4)                 2052
_________________________________________________________________
activation_3 (Activation)    (None, 4)                 0
=================================================================
Total params: 175,924
Trainable params: 175,924
Non-trainable params: 0
_________________________________________________________________
```

[60]:
```
# initial learning rate
init_lr = 5e-3
```

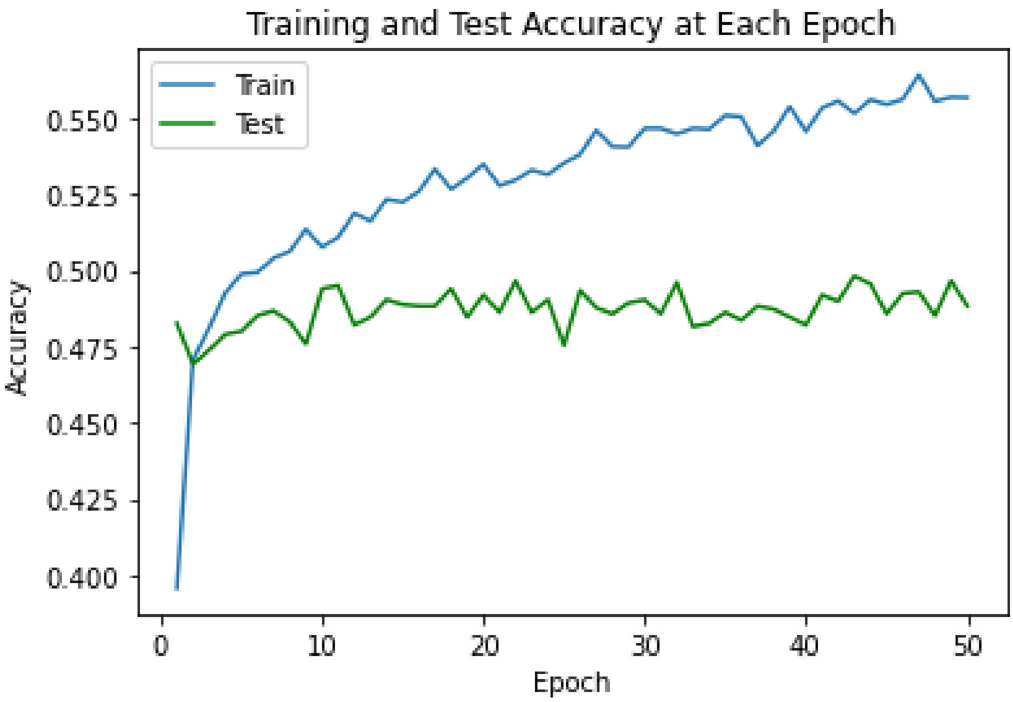
```
batch = 128
epoch = 50

# clear default graph
s = reset_tf_session()

# define the model
model = make_cnnmodel()

# prepare model for fitting
model.compile(
    loss='categorical_crossentropy',  # train 4-way classification
    optimizer=keras.optimizers.adamax(lr = init_lr),
    metrics=['accuracy']  # report accuracy during training
)
```

[61]:
```
# fit the model
start = time.time()
model.fit(
    X_tr, Y_tr,
    batch_size = batch,
    epochs = epoch,
    validation_data=(X_val, Y_val),
    shuffle=True,
    verbose=0,
    initial_epoch=0
)
end = time.time()
cnn_time = end - start # record the run time
```

[62]:
```
print('run time:', cnn_time)
```

```
run time: 62.177632093429565
```

[63]:
```
# plot training and validation accuracy

model_results = model.history.history
fig = plt.figure()
plt.plot(range(1,epoch+1), model_results['accuracy'], label='Train')
plt.plot(range(1,epoch+1), model_results['val_accuracy'], label='Test',␣
↪color='green')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Training and Test Accuracy at Each Epoch')
plt.show()
fig.savefig('Training and Test Accuracy at Each Epoch.png')
```



Sigmoid

[64]:
```
def make_cnnmodel_sig():
    """
    Define the model architecture.

    """
    model = Sequential()

    model.add(Conv1D(filters = 16, kernel_size = 3, padding = 'same',␣
↪input_shape=(1,72)))
    model.add(Activation('sigmoid'))
    model.add(Conv1D(filters = 32, kernel_size = 3, padding = 'same'))
    model.add(Activation('sigmoid'))
    model.add(Dropout(0.25))


    model.add(Flatten(input_shape=(1,72)))
    model.add(Dense(128, activation = 'sigmoid'))
    model.add(Dropout(0.25))
    model.add(Dense(256, activation = 'sigmoid'))
    model.add(Dropout(0.25))

    model.add(Dense(512, activation = 'sigmoid'))
```

```
    model.add(Dropout(0.25))
    model.add(Dense(4))
    model.add(Activation('softmax'))

    return model
```

[65]:
```
# initial learning rate
init_lr = 5e-3
batch = 128
epoch = 50

# clear default graph
s = reset_tf_session()

# define the model
model_1 = make_cnnmodel_sig()

# prepare model for fitting
model_1.compile(
    loss='categorical_crossentropy',  # train 4-way classification
    optimizer=keras.optimizers.adamax(lr = init_lr),  # for SGD
    metrics=['accuracy']  # report accuracy during training
)

# fit the model
start = time.time()
model_1.fit(
    X_tr, Y_tr,
    batch_size = batch,
    epochs = epoch,
    validation_data=(X_val, Y_val),
    shuffle=True,
    verbose=0,
    initial_epoch=0
)
end = time.time()
cnn_time_1 = end - start # record the run time
```

Tanh

[66]:
```
def make_cnnmodel_tanh():
    """
    Define the model architecture.

    """
    model = Sequential()
```

```
    model.add(Conv1D(filters = 16, kernel_size = 3, padding = 'same',␣
↪input_shape=(1,72)))
    model.add(Activation('tanh'))
    model.add(Conv1D(filters = 32, kernel_size = 3, padding = 'same'))
    model.add(Activation('tanh'))
    model.add(Dropout(0.25))


    model.add(Flatten(input_shape=(1,72)))
    model.add(Dense(128, activation = 'tanh'))
    model.add(Dropout(0.25))
    model.add(Dense(256, activation = 'tanh'))
    model.add(Dropout(0.25))
    model.add(Dense(512, activation = 'tanh'))
    model.add(Dropout(0.25))
    model.add(Dense(4))
    model.add(Activation('softmax'))

    return model
```

[67]:

```
# initial learning rate
init_lr = 5e-3
batch = 128
epoch = 50

# clear default graph
s = reset_tf_session()

# define the model
model_2 = make_cnnmodel_tanh()

# prepare model for fitting
model_2.compile(
    loss='categorical_crossentropy',  # train 4-way classification
    optimizer=keras.optimizers.adamax(lr = init_lr),  # for SGD
    metrics=['accuracy']  # report accuracy during training
)

# fit the model
start = time.time()
model_2.fit(
    X_tr, Y_tr,
    batch_size = batch,
    epochs = epoch,
    validation_data=(X_val, Y_val),
    shuffle=True,
    verbose=0,
```

```
    initial_epoch=0
)
end = time.time()
cnn_time_2 = end - start # record the run time
```

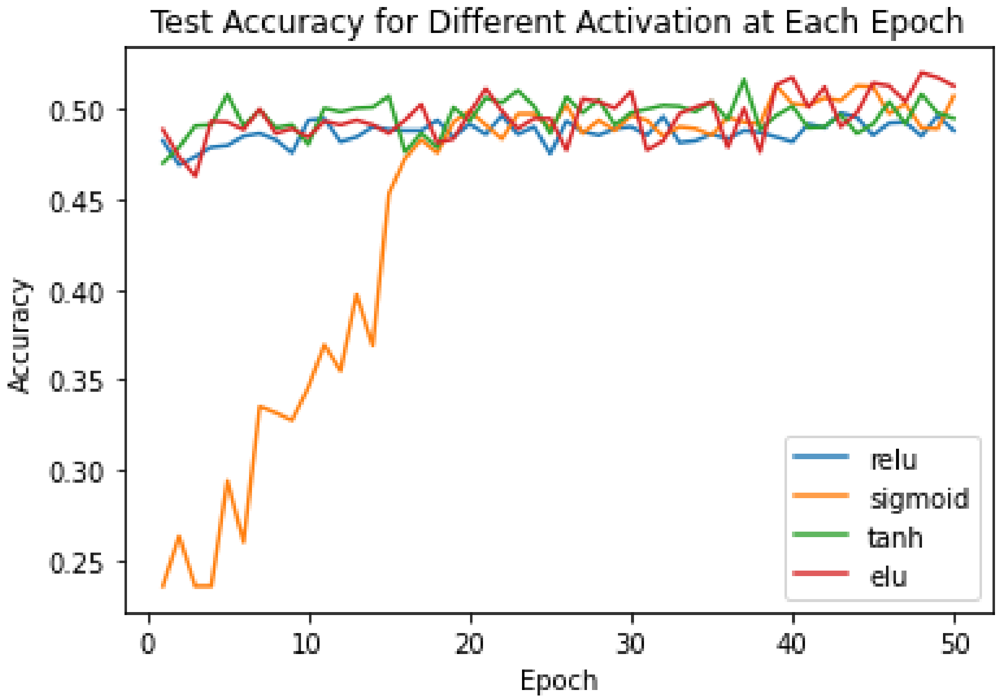
elu

[68]:
```
def make_cnnmodel_elu():
    """
    Define the model architecture.

    """
    model = Sequential()

    model.add(Conv1D(filters = 16, kernel_size = 3, padding = 'same',␣
 ↪input_shape=(1,72)))
    model.add(Activation('elu'))
    model.add(Conv1D(filters = 32, kernel_size = 3, padding = 'same'))
    model.add(Activation('elu'))
    model.add(Dropout(0.25))


    model.add(Flatten(input_shape=(1,72)))
    model.add(Dense(128, activation = 'elu'))
    model.add(Dropout(0.25))
    model.add(Dense(256, activation = 'elu'))
    model.add(Dropout(0.25))
    model.add(Dense(512, activation = 'elu'))
    model.add(Dropout(0.25))
    model.add(Dense(4))
    model.add(Activation('softmax'))

    return model
```

[69]:
```
# initial learning rate
init_lr = 5e-3
batch = 128
epoch = 50

# clear default graph
s = reset_tf_session()

# define the model
model_3 = make_cnnmodel_elu()

# prepare model for fitting
model_3.compile(
```

```
    loss='categorical_crossentropy',  # train 4-way classification
    optimizer=keras.optimizers.adamax(lr = init_lr),  # for SGD
    metrics=['accuracy']  # report accuracy during training
)

# fit the model
start = time.time()
model_3.fit(
    X_tr, Y_tr,
    batch_size = batch,
    epochs = epoch,
    validation_data=(X_val, Y_val),
    shuffle=True,
    verbose=0,
    initial_epoch=0
)
end = time.time()
cnn_time_3 = end - start # record the run time
```

[70]:
```
# plot the results for activation functions
results = model.history.history
results_1 = model_1.history.history
results_2 = model_2.history.history
results_3 = model_3.history.history
fig = plt.figure()
plt.plot(range(1,epoch+1), results['val_accuracy'], label='relu')
plt.plot(range(1,epoch+1), results_1['val_accuracy'], label='sigmoid')
plt.plot(range(1,epoch+1), results_2['val_accuracy'], label='tanh')
plt.plot(range(1,epoch+1), results_3['val_accuracy'], label='elu')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Test Accuracy for Different Activation at Each Epoch')
plt.show()
fig.savefig('Test Accuracy for Different Activation at Each Epoch.png')
```



[71]:
```
# total time for tuning
cnn_time + cnn_time_1 + cnn_time_2 + cnn_time_3
```

[71]: 272.16894793510437

2.2.3 Run the whole training set

[73]:
```
# preprocess the data
Y_cnn = Y_oh.to_numpy()
X_cnn = X.to_numpy()
# add 1 dim to the array
X_cnn=np.expand_dims(X_cnn,axis=1)
```

[75]:
```
# initial learning rate
init_lr = 5e-3
batch = 128
epoch = 50

# clear default graph
s = reset_tf_session()

# define the model
model = make_cnnmodel()
```

```python
# prepare model for fitting
model.compile(
    loss='categorical_crossentropy',  # train 4-way classification
    optimizer=keras.optimizers.adamax(lr = init_lr),
    metrics=['accuracy']  # report accuracy during training
)
# fit the model
start = time.time()
model.fit(
    X_cnn, Y_cnn,
    batch_size = batch,
    epochs = epoch,
    shuffle=True,
    verbose=0,
    initial_epoch=0
)
end = time.time()
cnn_time = end - start # record the run time

print('run time:', cnn_time)
```

```
run time: 69.4025490283966
```

3 Prediction

[80]:
```python
# use test data to make prediction
pred_rdf = pd.DataFrame({'price':clf_rdf.predict(X_test)})
ID = pd.DataFrame(ID)
pred = ID.join(pred_rdf,lsuffix='_caller', rsuffix='_other')
pred = pred.set_index('id')
display(pred)
```

```
       price
id
7715       2
13196      2
13194      2
4673       2
11325      1
...      ...
12921      3
7174       2
9240       3
11663      1
4513       1
```

```
[4149 rows x 1 columns]
```

[81]:
```
# save to csv file
pred.to_csv('pred.csv')
```

[ ]:
```

```