Wine Taste Forecaster

1. Domain Background

Wine traders, consumers, and producers spend millions of dollars to target a specific tasting profile. Experiments on making wine are tremendously difficult due to the number of factors affecting the wine taste including weather, soil, water, temperature and mineral compositions. Production of wine can take years and certain type of taste only appear after its maturity. This process may take up to 20 years. It is therefore important that the stakeholders know in advance what the wine will taste like.

This project aims to create a forecaster on wine taste and rating based on location, time, temperature, type of grapes used, and rainfall.

1.1 Taste and quality of wine

The result of this forecaster will be useful to winemakers who can set the price and release the wine based on the quality expected. Market speculator are able to forecast the quality of wine before they mature. Consumers are able to know which wine will have the characteristics that they desire.

1.2 Blending and import of grapes

Producers usually import grapes from different regions in order to match the taste profile that they desire. This predictor will give them an insight into the quality of the grapes in each region. This will be especially useful for producers in a non-traditional wine region who rely heavily on imports.

1.3 Maturity of wine

Ageing of wine is one of the most important decisions that all the stakeholders need to make. It is absolutely critical that wine is aged ideally. Ageing too long, the wine will turn sour. Ageing too little, the wine would have yet to reach the optimum maturity.

The decisions are traditionally made through tasting. However, this is an unreliable and expensive way to draw the conclusion. The result of this model will help the users to minimize cost and maximize profit.

2 Problem statement

The goal of the project is to create a forecaster which takes in information as listed in the input array and output a taste index, taste description, wine price, and wine score

2.1 Inputs

- Weather data
    - Average Temperature (daily)
        - Over the year before production
        - Over the year of production
    - Average Rainfall (daily)
        - Over the year before production
        - Over the year of production
- Location data
    - Country of production
    - Province of production
    - Vineyard name
    - GPS location of the vineyard
- Raw material data
    - Grape types

2.2 Outputs

- Wine Taste note rating array (to be used to display as a chart/paragraph with a webapp)
- Price
- Quality rating by Winemag

3 Solution Statement

The tasks involved are following.

1. Parse and clean all the training data
    a. Wine data is moved into a wine_df dataframe
    b. the latitude/longitude data is extracted using geopy and appended to wine_df.
    c. Year of production is parsed from the description text

    d. Temperature data downloaded from kaggle database
        i. Temp data is grouped by latitude/longitude and year of production.
        ii. Resultant df will have 365 columns of max/min temperature per day of each year for each latitude/longitude region.
        iii. Weather data is appended to each wine by closest location, year of production, year prior to production
        iv. Each weather data column will result in 730 additional column for each wine in wine_df (2 years of daily temp)
        v. Name column [tempi] where i is number of day from the date of origin (1st Jan of year before production)
2. Rainfall data is scraped from the worldclimatedata website[11] using a selenium script.
    a. Rainfall data is grouped by latitude/longitude and year of production.
    b. Resultant df will have 365 columns of rainfall per day of each year for each latitude/longitude region.
    c. Rainfall data is appended to each wine by closest location, year of production, year prior to production
    d. Each rainfall data column will result in 730 additional column for each wine in wine_df (2 years of daily rainfall)
    e. Name column [rainii] where i is number of day from the date of origin (1st Jan of year before production)
3. Tokenization and filter out unrelated keys
    a. In wine_df['description'] column, the description of wine has to be tokenized and counted for the key word of interest
        i. Use nltk library to filter out unrelated words, remove prefix ,and suffix
        ii. Tokenize the word with countVectorizer tokenizer
        iii. Use top 20-30 flavouring description that appear most frequently and count them against the description
        iv. Append taste note to wine_df as additional columns. Name the columns[taste index i] where i is the key.
4. One hot encode the columns [country, designation, province, region_1, region_2, variety, winery] replace null with zero. Then append this to wine_df.
5. Plot correlation and potentially remove some X column
6. Split data into train and test set using sklearn.train_test_split and add label to each column.
    a. Y columns = [price, point, taste index i]
    b. X columns = [tempi, raini, country, designation, province, region_1, region_2, variety, winery]
7. Create a model to train on the list
    a. Models in consideration for categorical Y [taste index]
        i. Pytorch
        ii. SKLearn
            1. SVC
            2. SVC linear

            3. K- nearest
            4. Random-forest
            5. Linear-regression
        iii. XGBoost
        iv. LightGBM
    b. Model for forecasting a variable include
        i. Pytorch
        ii. SKLearn
            1. Lasso
            2. Elasticnet
            3. Ridgeregression
        iii. XGBoost
        iv. LightGBM
    c. Determine which model would perform best
        i. For pytorch, test around 10 different configuration
    d. Potentially combine the model with ensemble
8. Create a RESTApi with API gateway and aws lambda (in future project)
9. Create a webapp to interact with the gateway and gather input-output (in future project)

4 Evaluation Metrics

- Taste note
    - F1 score

F1 score is a harmonic mean of precision and recall

$$F_1 = \left(\frac{2}{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}} \right) = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}$$

Where Precision = (TruePositive) / (TruePositive+FalsePositive)
Recall =(TruePositive)/(TruePositive+FalseNegative)

    - Justification
        - There is no indication that the data is balanced which means accuracy can be misleading.
        - Binary classification data
        - F1 is a harmonic mean of precision and recall which are both good indicators of the performance of the model

- Score/Price
    - Relative Absolute Error (RAE) and Mean Absolute Error (MAE)

- This is suitable since there are likely to be a lot of outliers in both wine score and wine price. We would like those to not affect the actual model too much.
- Wine price is affected strongly by other factors such as fashion, marketing, and general market conditions. It is expected that there will be a high number of outliers

5 Benchmark Model

I have found various projects that has similarities to this proposal.

1. Robinson, S., (2019)[4] research is using the same Kaggle data set to determine the price of wine. However, It is using the bag of word description to forecast the price and she is not sharing the end result so the model needs to be reproduced to calculate the evaluation metrics.
2. cortez, P., (2019)[8] aims to forecast wine price using weather data on a linear regression model and comes up with a matrix of price probability table for each vineyard.
3. Freecodecamp(2018)[6] article tries to understand what makes wine taste good based on its chemical characteristics using wine dataset from UCI (Uciedu, 2019).
4. Olivier goutay, (2018)[9] forecasts the 5 groups of wine quality based on its description. Which the author claim has over 97% efficiency using random forest classifier.

After reading all the articles, I come up with the following benchmarks

1. Wine description forecast should be better than a linear regression classifier using sklearn. (Model will be built as a benchmark)
2. Wine group of quality should be at least 97% accurate according to research 4.
3. Wine prices have MAE of 4 according to the 3 results presented in research 1.

6 Analysis

6.1 Data Exploration

1. Kaggle wine dataset which is cleaned was available so that was used instead of the original 150,929 instances. The tasting data set used is from the Kaggle wine dataset version 2[1] by the magazine Winemag[2].

a. Nans
    i. The data includes 129,908 instances. Upon inspection, the following Nan was found.

```
country            63
description         0
designation     37465
points              0
price            8996
province           63
region_1        21247
region_2        79460
title               0
variety             1
winery              0
dtype: int64
```

Fig 6.1.1 summary of nan data in the kaggle wine dataset

    ii. It was decided that designation, region1, and region2 can be combined and changed into numerical latitude and longitude numbers therefore the nans are ignored.
    iii. The price nan number of 8996 is an insignificant number compared to 130k in the dataset. Therefore the Nans are dropped.

b. Distribution
    i. following columns are categorical values [**country, description, designation, point, price, province, region_1, region_2, variety, winery**] with the unique categorical values in table 6.1.4.

	country	description	designation	province	region_1	region_2	taster_name	taster_twitter_handle	title	variety
count	129908	129971	92506	129908	108724	50511	103727	98758	129971	129970
unique	43	119955	37979	425	1229	17	19	15	118840	707

Fig 6.1.4 summary of count of non nan values and unique value of categorical values

    ii. **Price** is given in USD and has an extremely high positive skewness as seen in fig 6.1.2

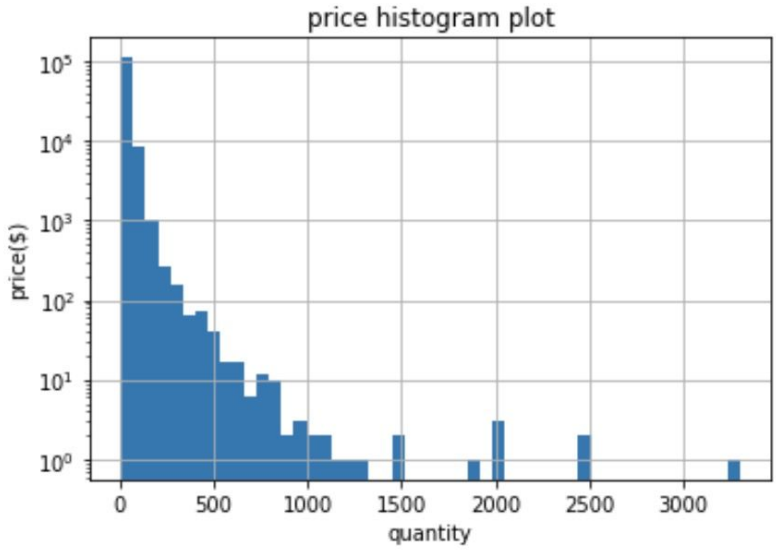
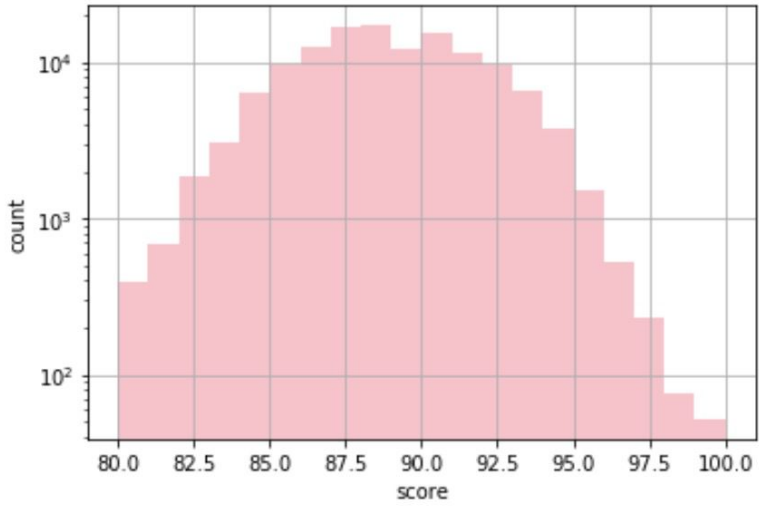


Fig 6.1.2 Price histogram plot

iii. The **wine score** is given by winemag between 0 and 100 according to winemag rating. With an average score of 88 points with a standard deviation of 3.



6.1.3 Distribution of wine score by winemag

2. Temperature data is from Kaggle Global Warming database by @berkeleyearth[3].
    a. For This dataset, there are 5 columns **[datatime, max daily temp, min daily temp, latitude, longitude,City]**
    b. The data contains records of daily temperature over 3448 cities from 1743 to 2013.

c. Temperature has a mean of 16.7 C and a standard deviation of 10.3 C. The data is heavily negatively skewed with a median of 26.5 C. This is shown in figure 6.1.4.

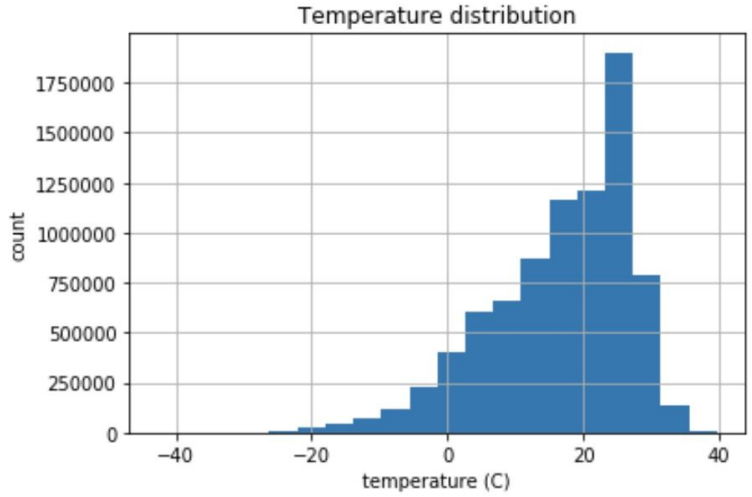


Figure 6.1.4 Distribution of temperature data

3. GPS coordinates is extracted from Geopy library 2018[10]
    a. The library is installed by pip3 using ```pip3 install geopy```
    b. Location of each city/vineyard are found using the nominatim API[12] which is an open sourced organization that collects and provides the GPS coordinates of every landmark.

6.2 Algorithm and techniques

There are many models in consideration for both ratio forecast and categorical forecast. The list of which are summarized in the table 6.2.1 and 6.2.2

6.2.1 Input data

Input data include the data of parsed cleaned temperature, latitude, longitude, and variety as shown in section 6.1. The input consists of 2440 columns with 27 ratio columns and 2413 categorial(one hot encoded) data.

6.2.2 Output data

The output is based on 2 different models; binary and ratio output model.

- Ratio output
    - Wine rating
    - Wine price
- Binary output
    - Taste note

- This is the top 100 most common comment in the taste description on wine_df

6.2.1 Neural Network

Neural network is a series of connected neurons(number storage) which are multiplied together according to their weights and biases and passed through an activation function in each layer.

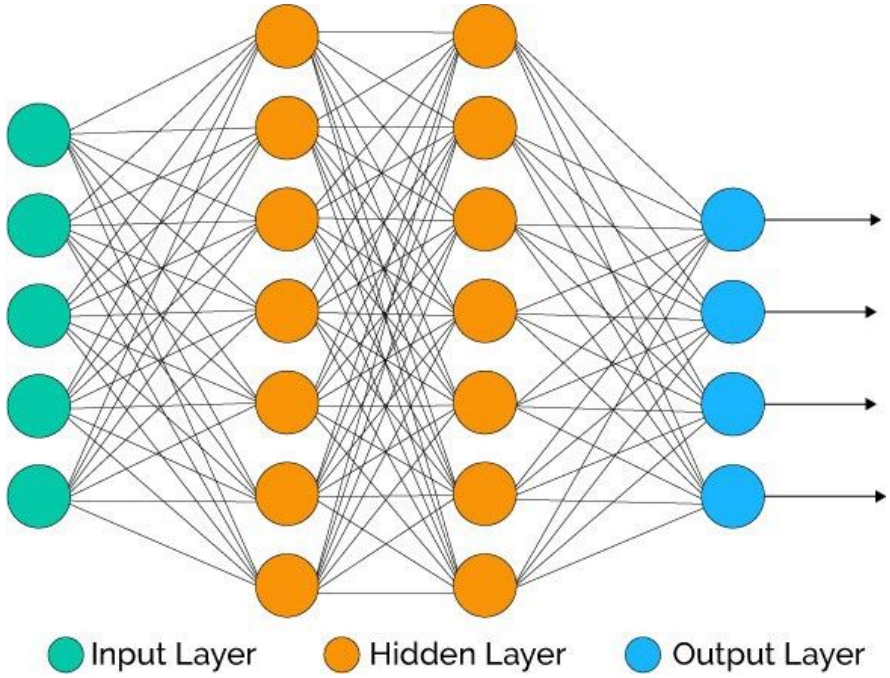


Fig 6.2.1.1 Diagram of a working neural network

Neural network has been proven to be a good models to forecast variable with a high number of input samples. Many models using the wine dataset such as Robinson, S., 2019[1] uses neural network as their primary model.

6.2.2 LightGBM model

LightGBM is a decision tree learning algorithm which is boolean function to forecast a variable based on a logical output of multiple inputs.

$$f : \{0,1\}^n \rightarrow \{0,1\}$$

Fig 6.2.2.1 Decision tree boolean function

LightGBM[17] library improves on traditional decision tree models' accuracy by growing leaf-wise(best-first). Choosing the leaf with max delta loss to grow. This greatly reduce the cost of training while improving accuracy of the model.

The training algorithm has been optimized by various methods including:

- Using histogram subtraction to speed up; only one leaf's histogram is required to be built which is then used for many neighbor histograms.
- Replacing continuous values with discrete bins; Uint8_t can be used instead of floating point. This eliminates the requirement to store sorting information.

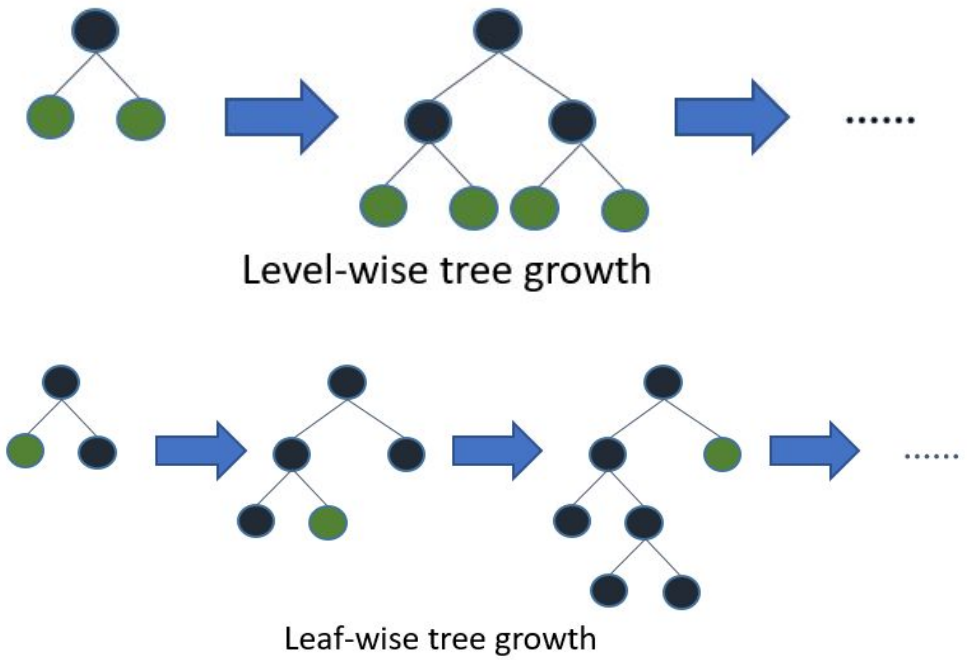


Fig 6.2.2.2 comparison between level wise and leaf-wise tree growth

LightGBM has been widely used in both categorical and ratio forecast[18].

6.2.3 XGBoost

XGBoost is one of the most popular tree boosting algorithms which is a combination of tree learning method (a simple base algorithm) with a gradient boosting algorithm(more advanced learning algorithm).

Gradient boosting minimizes an ensemble of the form in fig 6.2.3.1. Thus solving equation 6.2.3.2. With the method 6.2.3.3. This is similar to Adaboost algorithm.

Tree method is a simple model which has a limited power predicted power due to the limitation in its linearity. However, many of them can be combined together such as in Random forest model to produce a powerful predictive algorithm. The result of multiple tree will have stepping shape as seen in fig 6.2.3.4. It can be clearly seen than there is a lot of linearity in prediction shown by the flat area in the chart.

XGboost combines the benefit of both gradient boosting and tree method resulting in a prediction chart seen in fig 6.2.3.5. This clearly show a reduction in linearity while keeping the training time reasonably low. The method of iteration training for a gradient tree boosting method is shown in 6.2.3.6.[18]

$$f(x) = \sum_{m=0}^{M} f_m(x).$$

Fig 6.2.3.1 ensemble function for gradient boosting method

$$\{\hat{\theta}_m, \hat{\phi}_m\} = \underset{\{\theta_m, \phi_m\}}{\arg\min} \sum_{i=1}^{n} L(y_i, \hat{f}^{(m-1)}(x_i) + \theta_m \phi_m(x_i))$$

Fig 6.2.3.2 equation to achieve the minimum value for the ensemble function

Algorithm 1: Gradient boosting

Input : Data set $\mathcal{D}$.
A loss function L.
A base learner $\mathcal{L}_{\Phi}$.
The number of iterations M.
The learning rate η.

1 Initialize $\hat{f}^{(0)}(x) = \hat{f}_0(x) = \hat{\theta}_0 = \arg\min_{\theta} \sum_{i=1}^{n} L(y_i, \theta)$;

2 **for** $m = 1,2,..,M$ **do**

3 $\quad \hat{g}_m(x_i) = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}^{(m-1)}(x)}$;

4 $\quad \hat{\phi}_m = \arg\min_{\phi \in \Phi, \beta} \sum_{i=1}^{n} \left[\left(-\hat{g}_m(x_i)\right) - \beta\phi(x_i)\right]^2$;

5 $\quad \hat{\rho}_m = \arg\min_{\rho} \sum_{i=1}^{n} L(y_i, \hat{f}^{(m-1)}(x_i) + \rho\hat{\phi}_m(x_i))$;

6 $\quad \hat{f}_m(x) = \eta\hat{\rho}_m\hat{\phi}_m(x)$;

7 $\quad \hat{f}^{(m)}(x) = \hat{f}^{(m-1)}(x) + \hat{f}_m(x)$;

8 **end**

Output: $\hat{f}(x) \equiv \hat{f}^{(M)}(x) = \sum_{m=0}^{M} \hat{f}_m(x)$

Fig 6.2.3.3 function pseudocode for boosting algorithm

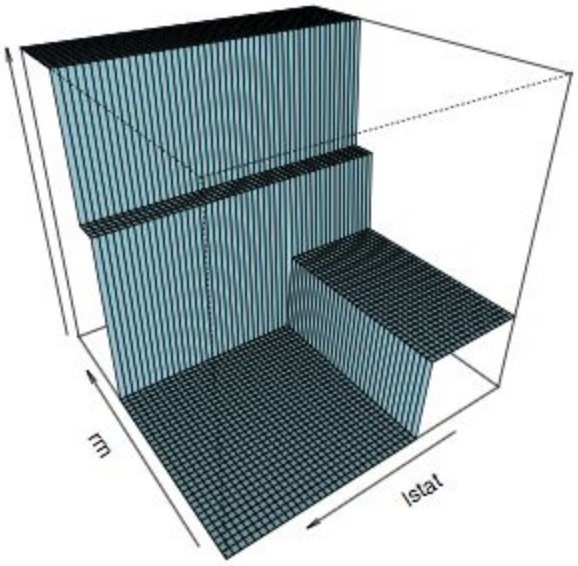


Fig 6.2.3.4 multiple decision tree prediction visualization chart

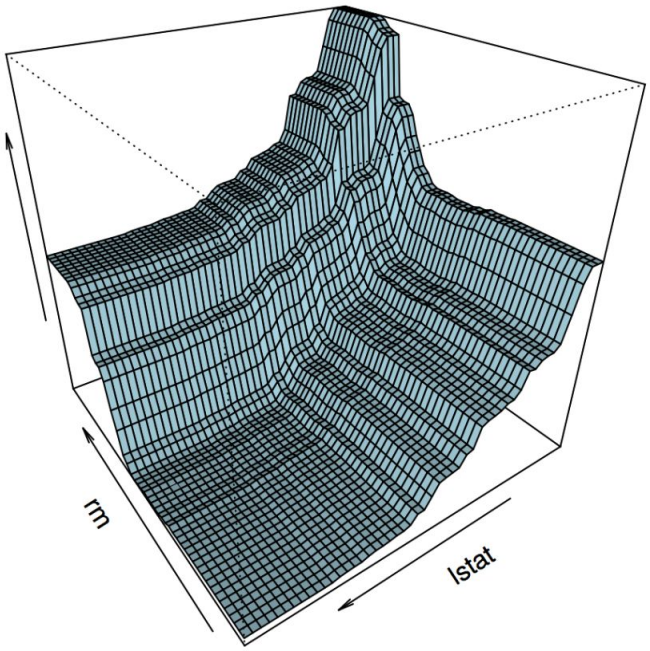


Fig 6.2.3.5 visualisation of a tree boosting algorithm

Algorithm 4: Gradient tree boosting

Input : Data set $\mathcal{D}$.
A loss function L.
The number of iterations M.
The learning rate η.
The number of terminal nodes T

Initialize $\hat{f}^{(0)}(x) = \hat{f}_0(x) = \hat{\theta}_0 = \arg\min_{\theta} \sum_{i=1}^{n} L(y_i, \theta)$;

for $m = 1,2,..,M$ **do**

  $\hat{g}_m(x_i) = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}^{(m-1)}(x)}$;

  Determine the structure $\{\hat{R}_{jm}\}_{j=1}^{T}$ by selecting splits which maximize $Gain = \frac{1}{2}\left[\frac{G_L^2}{n_L} + \frac{G_R^2}{n_R} - \frac{G_{jm}^2}{n_{jm}}\right]$;

  Determine the leaf weights $\{\hat{w}_{jm}\}_{j=1}^{T}$ for the learnt structure by $\hat{w}_{jm} = \arg\min_{w_j} \sum_{i \in \hat{I}_{jm}} L(y_i, \hat{f}^{(m-1)}(x_i) + w_j)$;

  $\hat{f}_m(x) = \eta \sum_{j=1}^{T} \hat{w}_{jm} \mathrm{I}(x_i \in \hat{R}_{jm})$;

  $\hat{f}^{(m)}(x) = \hat{f}^{(m-1)}(x) + \hat{f}_m(x)$;

end

Output: $\hat{f}(x) \equiv \hat{f}^{(M)}(x) = \sum_{m=0}^{M} \hat{f}_m(x)$

Fig 6.2.3.6 Gradient tree boosting algorithm

6.2.4 Ridge Regressor

Ridge regression is biased regression technique which is a good technique for analyzing multiple data which suffer from multicollinearity. Which is a non-linear relationship between data.

Ridge Regression Models can be written in the matrix form Y = XB + e where Y is the dependent variable, X represents the independent variables, B is the regression coefficients to be estimated, and e represents the errors are residuals. B is estimated by equation 6.2.4.1

$$\underline{\widetilde{\mathbf{B}}} = \left(\mathbf{R} + k\mathbf{I}\right)^{-1} \mathbf{X}' \underline{\mathbf{Y}}$$

Equation 6.2.4.1 estimation of a better regression coefficient where R is the correlation matrix of the variable.

Ridge regression has been shown to perform well in the wine rating estimation model

6.2.5 Other models

There are various other forecasting models which are available from Scikitlearn library. The summary of those are shown in tables 6.2.5.1 and 6.2.5.2. These other models can be used for comparison and benchmarking on how the selected model has performed.

model	Training parameters	architecture	Note
Neural network	Epoch Batch size Optimizer Learning rate Learning rate decay Dropout Momentum	Number of hidden layer Layer type (convolutional, fully connected, LSTM, or pooling)	
Random-forest	N_estimators criterion='gini', max_depth=2, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0, oob_score=False, random_state=0,	Random state	
XGBoost	Max_depth Eta objective		**XGBoost** is an optimized distributed gradient boosting library designed to be highly **efficient**, **flexible** and **portable**.XGBoost provides a parallel tree boosting.

LightGBM	Num_leaves Max_depth Leraning_rate N-estimator Subsample_for_bin Objective (binary, multiclass, lambdarank)		Optimized decision tree regressor. Grow trees leaf-wise
AdaboostClassifier	Base_estimator N_estimators (terminate after done n estimators) Algorithm Random state	Estimators (list of classifiers) Estimator weight	Combine different estimators together

Table 6.2.5.1 Classification model

model	Training parameters	architecture	Note
Neural network	Epoch Batch size Optimizer Learning rate Learning rate decay Dropout Momentum	Number of hidden layer Layer type (convolutional, fully connected, LSTM, or pooling)	Use multiple layers of neurons to create a complex predictor
Lasso	Alpha (panelty) Max_iter Tolorence Selection (random or cyclic) Random state	Number of iteration Parameter vector Sparse coefficient	Maximize objective function `(1 / (2 * n_samples)) * ||y - Xw||^2_2 + alpha * ||w||_1`
Elasticnet	Alpha (panelty) L1_ratio (mixing param between 0 and 1) Max_iter Tolorence Selection (random or cyclic)	Number of iteration Parameter vector Sparse coefficient	Maximize objective function `1 / (2 * n_samples) * ||y - Xw||^2_2 + alpha * l1_ratio * ||w||_1 + 0.5 * alpha * (1 - l1_ratio) * ||w||^2_2`

Ridgeregression	Alpha (panelty) Max_iter(maximum iteration) Tolorence (precision for solution) Solver (**{'auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag', 'saga'})**	Number of iteration Weight vectors	Maximize objective function `||y - Xw||^2_2 + alpha * ||w||^2_2`
AdaBoostRegressor	Base_estimator N_estimators (terminate after done n estimators) Learning_rate Loss function Random state	Estimator list Estimator weight	Combine different estimators together
XGBoost			
LightGBM	Num_leaves Max_depth Leraning_rate N-estimator Subsample_for_bin Objective (binary, multiclass, lambdarank)		Optimized decision tree regressor. Grow trees leaf-wise

Table 6.2.5.2 Ratio model

i.

6.3 Benchmark

7. Methodology

7.1 Data Preprocessing

7.1.1 The tasks involved are following.

1. Combine weather, wine_reviews, designation area, and locational data together.
    a. Download all the data
        i. Wine data is downloaded from kaggle using the Kaggle API then Unzipped into the ./data directory.

```
#download wine-reviews
kaggle.api.dataset_download_file('zynicide/wine-reviews','winemag-data-130k-v2.csv'
,path='./data',quiet=True,force=True)
!unzip './data/winemag-data-130k-v2.csv.zip' -d './data'
```

```
wine_df=pd.read_csv('./data/winemag-data-130k-v2.csv',index_col=0)
```

        ii. Temperature data is downloaded and saved into ./data directory.

```
#download temperature_data
kaggle.api.dataset_download_file('berkeleyearth/climate-change-earth-surface-temper
ature-data',

'GlobalLandTemperaturesByCity.csv',path='./data',quiet=True,force=True)
!unzip './data/GlobalLandTemperaturesByCity.csv.zip' -d './data'
```

    b. Add Latitude/Longitude data to the wine_df
        i. the latitude/longitude data is extracted from a description of location including country, city name.

```
## function for appending the coordinate
## construct a loop function
from geopy.geocoders import Nominatim
geolocator = Nominatim(user_agent="wine_app")

from geopy.extra.rate_limiter import RateLimiter
geocode = RateLimiter(geolocator.geocode, min_delay_seconds=1)
def get_location(Location_desc_array,country_array,cache_location='location_cache.csv'):
    ## get cache or create cache if cache doesnt exist
    if os.path.isfile(cache_location):
        cache = pd1.read_csv(cache_location,index_col=0)
    else:
        cache = pd1.DataFrame({'location_desc':[],'point':[]})
    location=[]
# loop through all the data and append the location
    for location_desc,country in zip(location_desc_array,country_array):
        if cache['location_desc'].isin([location_desc]).any():
            slice_of_cache = cache[cache['location_desc']==location_desc]
            location.append(slice_of_cache['point'][0])
            print('using cache')
        else:
            print('using api')
            response = geocode(location_desc)
```

```
            ## catch error if response is none
            if response == None:
                #if no response try using just the country name
                response = geocode(country)
                if response == None:
                    response = 'error'
                    print ('didnt find location')
                    print(location_desc)
                    point = response
                else:
                    point = response.point
            else:
                point = geocode(location_desc).point
            new_df= pd1.DataFrame({'location_desc':[location_desc],'point':[point]})
            cache = cache.append(new_df)
            location.append(point)
    cache.to_csv(cache_location)
    return location
```

c. Adding the temperature data to wine_df

i. By inspection of the dataset downloaded, the data needs to be pivoted to be called by location and data

	dt	AverageTemperature	City	Country	Latitude	Longitude
0	1743-11-01	6.068	Århus	Denmark	57.05N	10.33E
5	1744-04-01	5.788	Århus	Denmark	57.05N	10.33E
6	1744-05-01	10.644	Århus	Denmark	57.05N	10.33E
7	1744-06-01	14.051	Århus	Denmark	57.05N	10.33E
8	1744-07-01	16.082	Århus	Denmark	57.05N	10.33E

ii. The code below is used to create a pivot table of temp_df; merging latitude and longitude.

```
#create an index name combining both the latitude and longitude data
temp_df['location_coordinate']= temp_df['Latitude']+' ' +temp_df['Longitude']
#Remove duplicates
temp_df=temp_df.drop_duplicates(subset=['location_coordinate','dt'])
#set appropriate index
temp_df=temp_df.set_index(['location_coordinate'])
# convert dt datatype from string to datetime
temp_df['dt']=pd.to_datetime(temp_df['dt'],format='%Y-%m-%d')
# pivot the temp_df
pivot_temp_df=temp_df.pivot(columns='dt'
                            ,values='AverageTemperature')
```

iii. Year of production is parsed from the description text

```
wine_df['year']=wine_df['title'].str.extract(r'(?P<digit>(?:\s)(?:19|20)\d\d(?:\s))
```

```
', expand=False)
```

iv. Find location of shortest distance between wine and weather data for each row in wine_df

```
from geopy import distance
from geopy.point import Point
unique_points=pd.DataFrame(list(temp_df.index.unique()))
def compare_distance(point1):
    return lambda x: distance.distance(point1,x).km
def find_minimum_dist(point_1,cache_location='./closest_location_cache.cache'):
    ## get cache file
    if os.path.isfile(cache_location):
        cache = pd1.read_csv(cache_location,index_col=0)
    else:
        cache = pd1.DataFrame({'point_1':[],'closest_point':[],'distance':[]})

    ## check if point is in the cache
    if cache['point_1'].isin([point_1]).any():
#         print('using cache')
        closest_point = cache[cache['point_1']==point_1]['closest_point'][0]
        closest_distance = cache[cache['point_1']==point_1]['distance'][0]
    else:
#         print('using apply')
        try:
            distance_array = unique_points[0].apply(compare_distance(point_1))
            closest_point = unique_points.loc[distance_array.idxmin()][0]
            closest_distance = distance_array[0].min()
        except ValueError:
            distance_array = 0
            closest_point = 0
            closest_distance = 0
            print('value error')
        ## save modified cache
        new_df=
pd1.DataFrame({'point_1':[point_1],'closest_point':[closest_point],'distance':[closest_dista
nce]})
        cache = cache.append(new_df)
        cache.to_csv(cache_location)
    return closest_point
wine_df['closest_location'] = wine_test['location'].apply(find_minimum_dist)
```

v. Temperature is appended to the wine_df based on the closest location to the temperature

```
def select_and_append_temp_loc(df):
    temp_new_df = pd.DataFrame()
    initial_series =
pivot_temp_df.filter(regex=r'(?P<digit>^{}|{})'.format(int(df.loc[0,'year']),int(df.loc[0,'y
```

```
ear'])-1)).loc[df.loc[0,'closest_location']].reset_index(drop=True)
    initial_series.index= initial_series.index.map(map_dict)
    temp_new_df = temp_new_df.append(initial_series)
    for i in range (len(df)):
        print(i)
        year = int(df.loc[i,'year'])
        try:
            if (df.loc[i,'closest_location'] == '0') or year == 0:
                if year == 0:
                    year = 1990
                print('empty closest location')

new_series=pivot_temp_df.filter(regex=r'(?P<digit>^{}|{})'.format(year,year-1)).loc['39.38N
8.32E']
            else:

new_series=pivot_temp_df.filter(regex=r'(?P<digit>^{}|{})'.format(year,year-1)).loc[df.loc[i
,'closest_location']]
            new_series=new_series.reset_index(drop=True)
            new_series.index=new_series.index.map(map_dict)
        except:
            print('error')
            new_series=pd.Series({0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9:
0, 10: 0, 11: 0, 12: 0, 13: 0, 14: 0, 15: 0, 16: 0, 17: 0, 18: 0, 19: 0, 20: 0, 21: 0, 22:
0, 23: 0})
            new_series=new_series.reset_index(drop=True)
            new_series.index=new_series.index.map(map_dict)
        temp_new_df.loc[i]=new_series
    temp_new_df.drop('39.38N 8.32E',inplace=True)
    return df.join(temp_new_df,how='outer')
processed_wine_df=select_and_append_temp_loc(wine_df)
```

vi. Wine data with year == 0 (without year data) and unnecessary columns including `['taster_name','taster_twitter_handle','closest_location']` are removed

```
wine_df=wine_df[wine_df['year']!=0]
# drop columns that we no longer need
wine_df=wine_df.drop(['taster_name','taster_twitter_handle','closest_location'],axis=1)
```

d. Append area described by wine designation to wine_df

i. Download and parse necessary appellation data from the wiki database

```
designation_string=
'''Ajaccio    Corsica 1984    Before 1984 a designation within Corse or Vin du Corse under
the alternative names Ajaccio or Coteaux d'Ajaccio
Aloxe-Corton  Burgundy      1938
Alsace Alsace 1945
Alsace Grand Cru     Alsace 1975
Anjou  Loire  1936
```

```
Anjou-Coteaux de la Loire    Loire  1946   ...
Vosne-Romanée  Burgundy      1936
Vougeot Burgundy       1936
Vouvray Loire  1936   '''

designation_df=pd.DataFrame([items.split('\t') for items in
designation_string.split('\n')]).rename({0:'designation',1:'area'},axis=1)[['designation','a
rea']]
```

ii. Use library Fuzzywuzzy[14] to match the closest description to the area

```
from fuzzywuzzy import fuzz,process
def get_ratio(row):
    name = row['designation']
    result=designation_df.iloc[designation_df['designation'].apply(lambda row:
fuzz.token_set_ratio(name,row)).idxmax(),1]
    return result
wine_df['area']=vintage_processed_wine_clean.apply(get_ratio,axis=1)
```

e. Get latitude and longitude in float values

```
import geopy
def get_latitude_longitude(row):
    print(row)
    try:
        location_sample=geopy.location.Location(point=row)
        return
pd.Series({"latitude":location_sample.latitude,"longitude":location_sample.longitude})
    except ValueError:
        return pd.Series({"latitude":0,"longitude":0})
wine_df = pd.concat([wine_df,
          wine_df['location'].apply(get_latitude_longitude)],
          axis=1)
```

2. Parse text description, tokenize and one hot encode.

a. Use library nltk[15] to create dictionary of most common words ignoring following words `['wine','drink','note','give','well','year','hint','great','nose','still','tight','open','yellow','like','end','pair','structur','dri']`

```
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import *

import re
from bs4 import BeautifulSoup

def review_to_words(review):
    nltk.download("stopwords", quiet=True)
    stemmer = PorterStemmer()
```

```
    text = BeautifulSoup(review, "html.parser").get_text() # Remove HTML tags
    text = re.sub(r"[^a-zA-Z0-9]", " ", text.lower()) # Convert to lower case
    words = text.split() # Split string into words
    words = [w for w in words if w not in stopwords.words("english")] # Remove stopwords
    words = [PorterStemmer().stem(w) for w in words] # stem

    return words
def build_dict(data, vocab_size = 100, ignore=[]):
    """Construct and return a dictionary mapping each of the most frequently appearing words
to a unique integer."""

    word_count = {}
    for sentence in data:
        for word in sentence:
            if word in word_count.keys():
                if len(word) > 2 and word not in ignore:
                    word_count[word] += 1
            else:
                word_count[word] = 0

    sorted_words = list(dict(sorted(word_count.items(),key=lambda
x:x[1],reverse=True)).keys())

    word_dict = {}
    for idx, word in enumerate(sorted_words[:vocab_size - 2]):
        word_dict[word] = idx + 2                               # 'infrequent' labels
    return word_dict
word_token_sample=vintage_processed_wine_clean['description'].apply(review_to_words)
word_dict=build_dict(word_token_sample, vocab_size =
100,ignore=['wine','drink','note','give','well','year','hint','great','nose','still','tight'
,'open','yellow','like','end','pair','structur','dri'])
```

b. Count and add to wine_df

```
def convert_and_pad(word_dict, sentence, pad=500):
    NOWORD = 0 # We will use 0 to represent the 'no word' category
    INFREQ = 1 # and we use 1 to represent the infrequent words, i.e.,
words not appearing in word_dict
    tokenized_sentence= review_to_words(sentence)
    key_count = pd.Series()
    for word in tokenized_sentence:
        if word in word_dict.keys():
            if word in key_count.keys():
                key_count[word] += 1
            else:
                key_count[word] = 1
    return key_count
```

```
# unit test parsing the dict and merge index
wine_df= wine_df.merge(wine_df['description'].apply(lambda x:
convert_and_pad(word_dict,x)).add_prefix('taste_'),left_index=True,
right_index=True)
```

c. Remove unnecessary columns and fillna with 0

```
wine_df=wine_df.drop(['description','title','Location_description'],axis=1)
.fillna(0)
```

3. One hot encoding

One hot encode the columns [country, designation, province, region_1, region_2, variety, winery] replace null with zero. Then append this to wine_df.

```
def one_hot_encode(df,columns_for_one_hot):
    '''One hot encode the columns for feeding into the model
    :df: Dataframe of the data
    :columns_for_one_hot: array name of columns to input
    :return: Dataframe of the one hot encoded data with prefix = columns_for_one_hot
    '''
    for column_for_one_hot in columns_for_one_hot:
        one_hot = pd.get_dummies(df[column_for_one_hot],prefix=column_for_one_hot)
        # Drop column B as it is now encoded
        df = df.drop(column_for_one_hot,axis = 1)
        # Join the encoded df
        df = df.join(one_hot,rsuffix=column_for_one_hot)
    return df
```

4. Data Visualization

d. categorical data

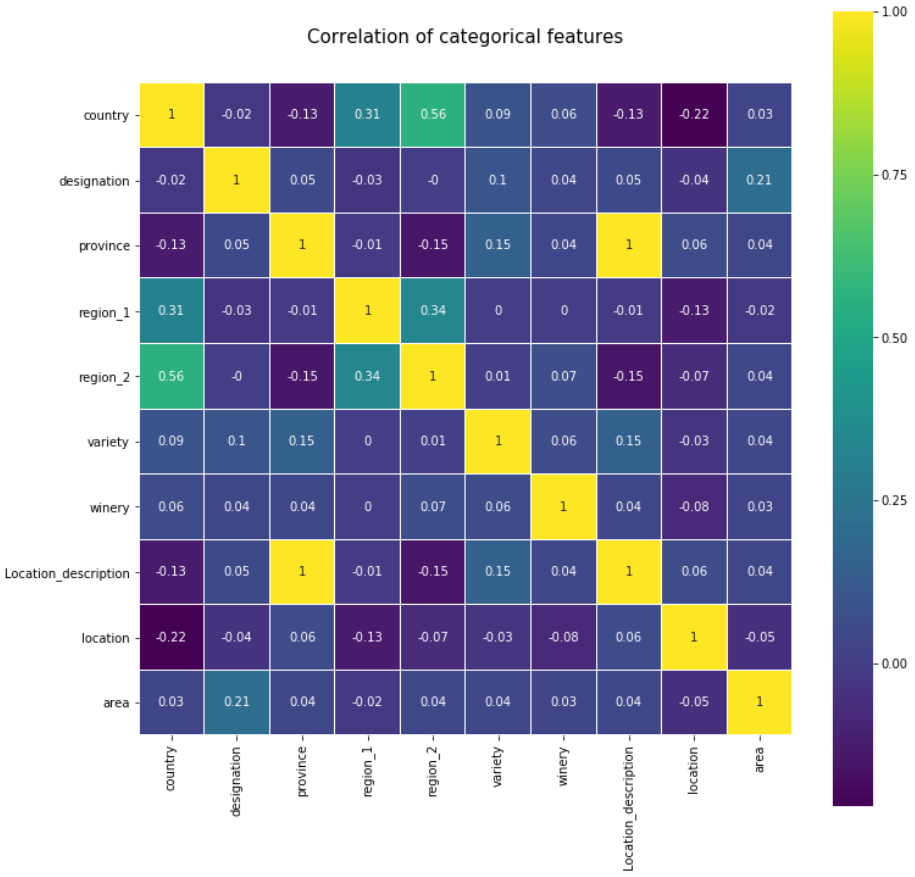
i. Check the unique data

It has been found that the count of designation, region1, winery has more than 1000 unique data. Therefore they should be removed

	country	designation	province	region_1	region_2	variety	winery	Location_description	location	area
count	125345	125345	125345	125345	125345	125345	125345	125345	125345	125345
unique	44	36089	420	1218	18	699	16346	420	352	17
top	US	0	California	0	0	Pinot Noir	Wines & Winemakers	California US	36 42m 5.26716s N, 118 45m 21.5906s W	Corsica
freq	53585	37160	35635	20693	75589	13126	221	35635	35635	39606

ii. Plot categorical data correlation

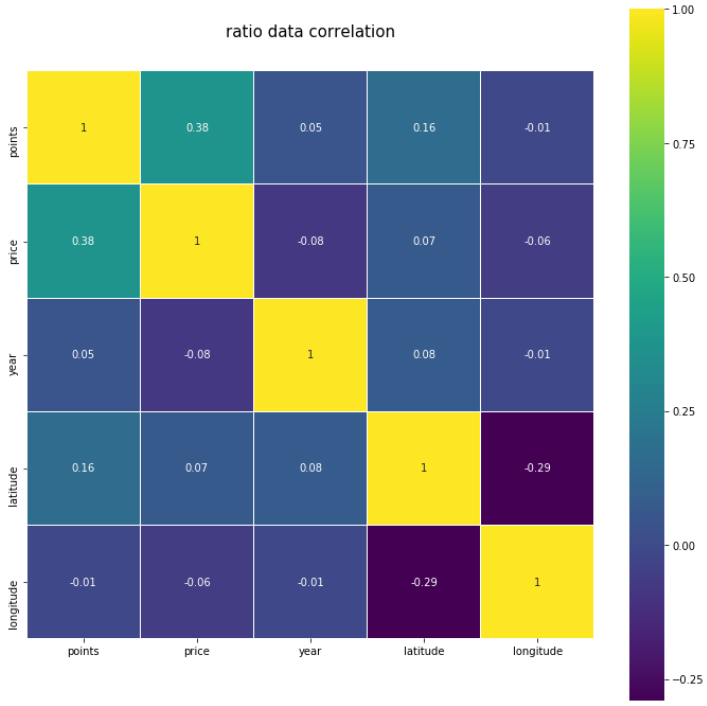
Only location description and province have a high correlation. Therefore province should be removed.



e. Ratio data

i. Plot correlation

All of the data have very low correlation to each other therefore they are all useful.



7.2 Implementation

7.2.1 Pre-process and split data into train/test set

- Remove non-useful columns and one hot encode categorical columns

```
to_encode=['area','country','province','region_1','region_2','variety']
to_drop=['designation','description','title','Location_description','location','winery']
df=df.drop(to_drop,axis=1)
df=one_hot_encode(df,to_encode)
```

- Split data into train and test set using sklearn.train_test_split and add label to each column.
    a. Firstly remove the columns of categorical output which was marked as starting with a prefix ‘taste_’. This was done using pandas regex filter.
    b. The data are then scaled to have values between 0 and 1 using sklearn minmaxscaler
    c. Y columns = [price, point]
    d. X columns = [temp, rain, country, designation, province, region_1, region_2, variety, winery]

```
ycol=['points','price']
regex_out_taste='^(?!taste)'
```

```
X = df.filter(regex=regex_out_taste).drop(ycol, axis=1) # remove the Y categorical
Y = dfg[ycol]              # Response / Target Variable
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
X=scaler.fit_transform(X)
Y=scaler.fit_transform(Y)
# print(Y)

print(X.shape, Y.shape)

# Split training set so that we validate on 20% of the data
# Note that our algorithms will never have seen the validation

np.random.seed(5875) # set random seed for reproducibility

from sklearn.model_selection import train_test_split

X_train, X_val, Y_train, Y_val = \
                train_test_split(X, Y, test_size=0.2)
```

7.2.2 Ratio model

Models that we use to test for the ratio model includes Neural Network, XGBoost, Lasso, Elasticnet, Ridgeregression, and LightGBM.

1. Input:

temperature, latitude, longitude, and variety

2. Output:

Wine rating, wine price.

3. Models

Neural network

A prototype model was built and tested using Keras library. There are many companies

- First test with NN model
    a. This is done using a keras library[16] with tensorflow backend
    b. The loss function used is mean_squared_error
    c. The matrix for verification is also mean_squared_error
    d. Optimizer used is adam with the default setting

e. The neural network has 1068:1000:500:2 nodes. The characteristics is summarized below

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_69 (Dense)             (None, 2000)              2334000
_________________________________________________________________
dropout_52 (Dropout)         (None, 2000)              0
_________________________________________________________________
dense_70 (Dense)             (None, 1000)              2001000
_________________________________________________________________
dropout_53 (Dropout)         (None, 1000)              0
_________________________________________________________________
dense_71 (Dense)             (None, 500)               500500
_________________________________________________________________
dropout_54 (Dropout)         (None, 500)               0
_________________________________________________________________
dense_72 (Dense)             (None, 2)                 1002
=================================================================
Total params: 4,836,502
Trainable params: 4,836,502
Non-trainable params: 0
_________________________________________________________________
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras import optimizers
# Y_train = to_categorical(Y_train)[:,0].squeeze()
model = Sequential()
model.add( Dense(units=2000,kernel_initializer='random_normal' ,activation='relu', input_shape=(1068,)
))
model.add( Dropout(0.5))
model.add( Dense(units=1000, activation='relu'))
model.add( Dropout(0.5))
model.add( Dense(units=500, activation='relu'))
model.add( Dropout(0.5))
model.add( Dense(units=2, activation='tanh') )
optimizer = optimizers.SGD(lr=0.0001, momentum=0.01, decay=0.01, nesterov=False)
model.compile(optimizer='adam',
              loss='mean_squared_error',
              metrics=['mean_squared_error'])
model.fit(X_train, Y_train, epochs = 20, batch_size= 30)
print('mean_squared error is:',np.round(model.evaluate(X_val,Y_val,batch_size=50,verbose=True)[1],2))
print(model.predict(X[40:50,:],batch_size=1,verbose=1)*100)
print(Y[40:50,:]*100)
```

The model performed with the mean squared error of 0.0081

Lasso

```
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
#lasso model
lasso = Lasso()   # instantiate
lasso.fit(X_train, Y_train)
y_pred= lasso.predict(X_val)
evaluate(y_pred,Y_val,y_col)
```

```
ycol is: ['points', 'price']
mean squared error is: [0.023469, 0.000141]
mean absolute error is: [0.125139, 0.006185]
median_absolute_error is: [0.124262, 0.004565]
```

Elasticnet

```
from sklearn.linear_model import ElasticNet
from sklearn.metrics import mean_squared_error
#elasticnet model
elas = ElasticNet()   # instantiate
elas.fit(X_train, Y_train)
y_pred = elas.predict(X_val)
evaluate(y_pred,Y_val,y_col)
```

```
ycol is: ['points', 'price']
mean squared error is: [0.023469, 0.000141]
mean absolute error is: [0.125139, 0.006185]
median_absolute_error is: [0.124262, 0.004565]
```

Ridge

```
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
#ridge model
ridge = Ridge()   # instantiate
ridge.fit(X_train, Y_train)                          # fit
y_pred = ridge.predict(X_val)
evaluate(y_pred,Y_val,y_col)
```

```
ycol is: ['points', 'price']
mean squared error is: [0.015954, 0.000108]
mean absolute error is: [0.100137, 0.004624]
median_absolute_error is: [0.083973, 0.002749]
```

LightGM

```
from lightgbm.sklearn import LGBMRegressor
from sklearn.multioutput import MultiOutputRegressor
lgbm = LGBMRegressor()           # create
lgbm_regr= MultiOutputRegressor(lgbm)
lgbm_regr.fit(X_train, Y_train)            # train
y_pred=lgbm_regr.predict(X_val)
evaluate(y_pred,Y_val,['points', 'price'])
```

```
ycol is: ['points', 'price']
mean squared error is: [0.015818, 0.000108]
mean absolute error is: [0.100263, 0.004548]
median_absolute_error is: [0.085476, 0.0026]
```

AdaBoost

```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.multioutput import MultiOutputRegressor
adab = AdaBoostRegressor()           # create
adab_regr= MultiOutputRegressor(adab)
adab_regr.fit(X_train, Y_train)            # train
y_pred=adab_regr.predict(X_val)
evaluate(y_pred,Y_val,['points', 'price'])
```

```
ycol is: ['points', 'price']
mean squared error is: [0.021732, 0.000143]
mean absolute error is: [0.120232, 0.00645]
median_absolute_error is: [0.105155, 0.005121]
```

LinearRegression (benchmark)

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
#linear regression
lr = LinearRegression()   # instantiate
lr.fit(X_train, Y_train)                        # fit
y_pred = lr.predict(X_val)
evaluate(y_pred,Y_val,y_col)
```

```
ycol is: ['points', 'price']
mean squared error is: [4317400221377447.0, 3714676496054658.5]
mean absolute error is: [878839.840054, 815193.200496]
median_absolute_error is: [0.084036, 0.002775]
```

4. Ratio Model comparison summary

All models perform satisfactorily with a reasonably low mean squared error, mean absolute error and, median absolute error. For the point prediction, it has been found that RidgeRegressor, LGBMRegressor, and neural network have the best performance. This is similar for the Price predictor except for neural network which performed relatively poorly in the price task. This may be attributed to overfitting due to a large irregularity in the price of wine.

Although the best model so far has been LGBMRegressor and Ridge, neural network usually improve with adjustment in hyperparameter including the layer configuration, optimizer, batch size, epoch, and number of hidden layers.

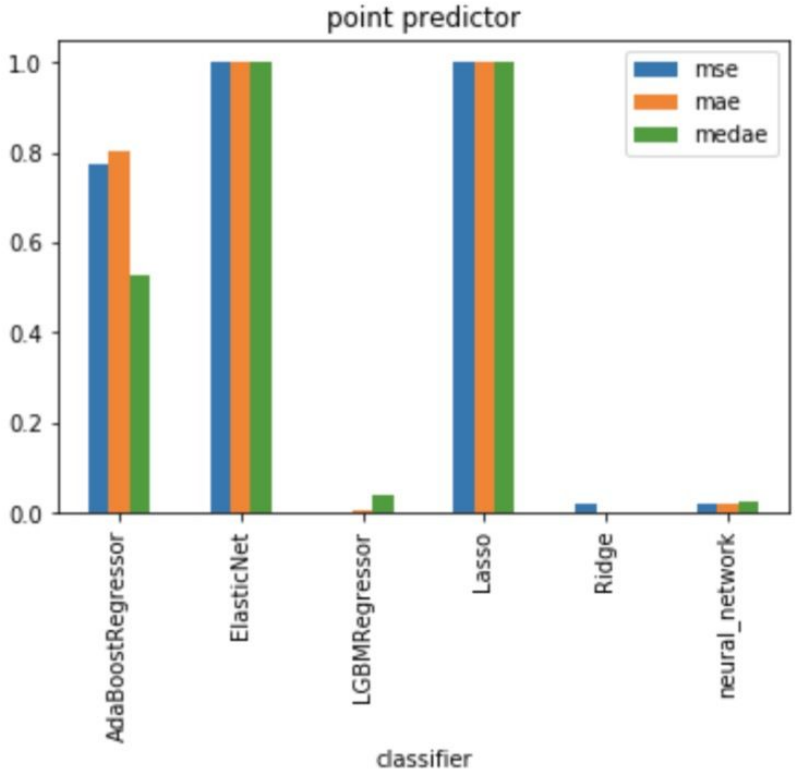


Fig 7.2.2.1 Points prediction summary

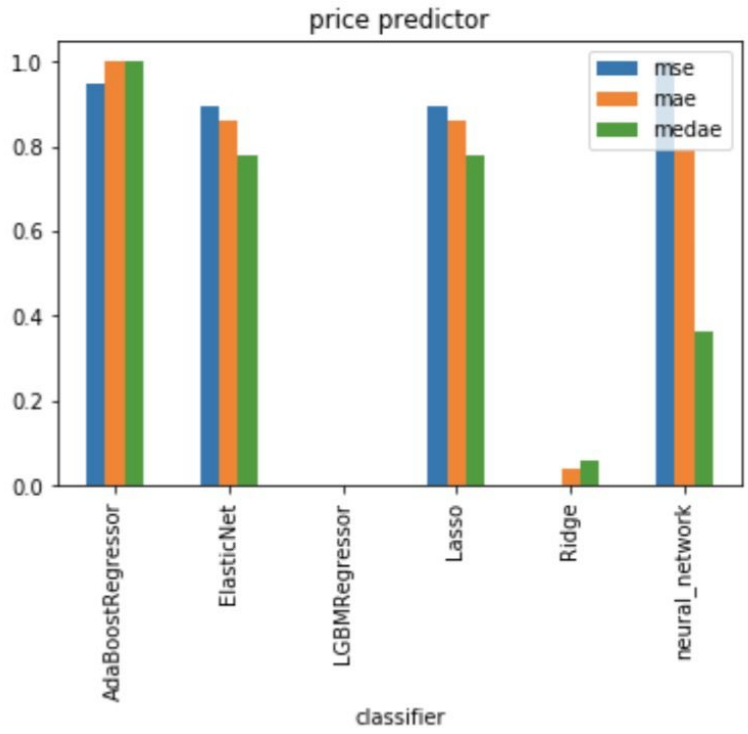


Fig 7.2.2.2 Price prediction summary

7.2.3 Categorical model

1. Input:

temperature, latitude, longitude, and variety

2. Output:

Taste notes as 98 array of most common keywords

Models

Models to try include neural network (Keras, Pytorch), SVC, SVC linear, K-nearest, Random-forest, Linear-regression, XGBoost, LightGBM

Dimensional reduction
It appears that many models such as K-nearest does not perform well with a large number of feature input layer. Therefore PCA is used for dimensional reduction.

Get Explained variance

```
# load decomposition to do PCA analysis with sklearn
from sklearn import decomposition
pca = decomposition.PCA(n_components=40)
pc = pca.fit_transform(X)
```

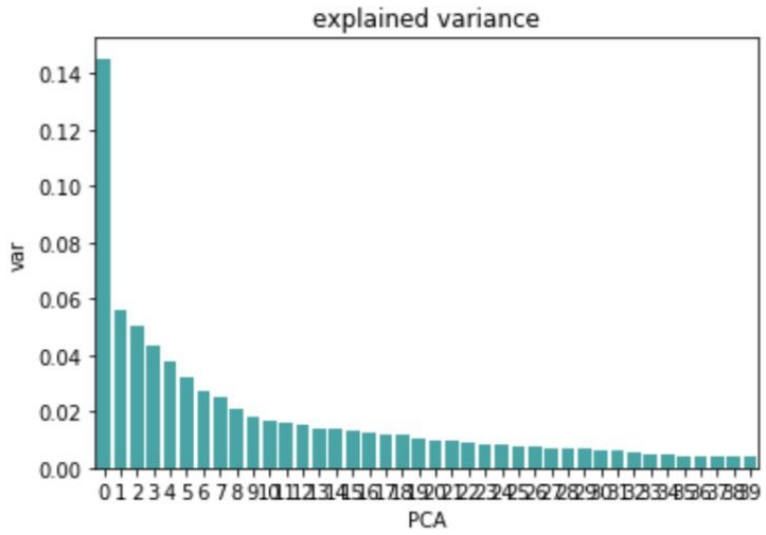


Fig 7.2.3.1 explained variance of pca dimensions

Select the top components that accounts to 70% of explained variance.

```
1  cum=pd.Series(pca.explained_variance_ratio_).cumsum()
2  cum
3  cum[cum<0.7].shape
```

```
(34,)
```

Fig 7.2.3.2 number of PCA components that contribute to 70% of explained variance

Neural Network

```
## neural network
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(1000,input_shape=[34,],activation='relu'))
model.add(Dense(500,activation='relu'))
model.add(Dense(98,activation='softmax'))
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
model.fit(X_train,Y_train,batch_size=100,epochs=10,verbose=1)

_,acc = model.evaluate(X_val,Y_val,verbose=0)
print('Accuracy',np.round(acc*100),'%')
y_pred=model.predict(X_val)
evaluate(y_pred.round(),Y_val.astype('int'))
```

```
f1_score is: 0.293169
accuracy_score is: 0.000199
```

GaussianNB

```
from sklearn.naive_bayes import GaussianNB # Gaussian Naive Bays
from sklearn.multioutput import MultiOutputClassifier
logreg = GaussianNB()           # create
regr= MultiOutputClassifier(logreg)
regr.fit(X_train, Y_train.astype('int'))            # train
y_pred=regr.predict(X_val)
evaluate(y_pred,Y_val.astype('int'))
```

```
f1_score is: 0.281375
accuracy_score is: 8e-06
```

Perceptron

```
from sklearn.linear_model import Perceptron
from sklearn.multioutput import MultiOutputClassifier
logreg = Perceptron()           # create
regr= MultiOutputClassifier(logreg)
regr.fit(X_train, Y_train.astype('int'))            # train
y_pred=regr.predict(X_val)
evaluate(y_pred,Y_val.astype('int'))
```

```
f1_score is: 0.230616
accuracy_score is: 0.0
```

DecisionTree

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.multioutput import MultiOutputClassifier
logreg = DecisionTreeClassifier()           # create
regr= MultiOutputClassifier(logreg)
regr.fit(X_train, Y_train.astype('int'))            # train
y_pred=regr.predict(X_val)
evaluate(y_pred,Y_val.astype('int'))
```

```
f1_score is: 0.245948
accuracy_score is: 0.001346
```

SVC

```
from sklearn.svm import SVC
from sklearn.multioutput import MultiOutputClassifier
logreg = SVC()            # create
regr= MultiOutputClassifier(logreg)
regr.fit(X_train, Y_train.astype('int'))            # t
y_pred=regr.predict(X_val)
evaluate(y_pred,Y_val.astype('int'))
```

```
f1_score is: 0.167129
accuracy_score is: 0.0
```

SVC linear

```
from sklearn.svm import LinearSVC
from sklearn.multioutput import MultiOutputClassifier
logreg = LinearSVC()          # create
regr= MultiOutputClassifier(logreg)
regr.fit(X_train, Y_train.astype('int'))            # train
y_pred=regr.predict(X_val)
evaluate(y_pred,Y_val.astype('int'))
```

```
f1_score is: 0.186769
accuracy_score is: 0.0
```

K-nearest

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.multioutput import MultiOutputClassifier
logreg = KNeighborsClassifier()           # create
regr= MultiOutputClassifier(logreg,6)
regr.fit(X_train, Y_train.astype('int'))            # train
y_pred=regr.predict(X_val)
evaluate(y_pred,Y_val.astype('int'))
```

```
f1_score is: 0.22877
accuracy_score is: 4e-05
```

Random-forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.multioutput import MultiOutputClassifier
logreg = RandomForestClassifier()           # create
regr= MultiOutputClassifier(logreg)
regr.fit(X_train, Y_train.astype('int'))            # train
y_pred=regr.predict(X_val)
evaluate(y_pred,Y_val.astype('int'))
```

```
f1_score is: 0.232676
accuracy_score is: 0.000363
```

Logistic-Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.multioutput import MultiOutputClassifier
logreg = LogisticRegression(solver='newton-cg')
regr= MultiOutputClassifier(logreg)
regr.fit(X_train, Y_train.astype('int'))            #
y_pred=regr.predict(X_val)
evaluate(y_pred,Y_val.astype('int'))

```

```
f1_score is: 0.244338
accuracy_score is: 0.000113
```

XGBoost

```
import xgboost as xgb
from sklearn.multioutput import MultiOutputClassifier
gbm = xgb.XGBClassifier(max_depth=3, n_estimators=300, learning_rate=0.05)
regr= MultiOutputClassifier(gbm)
regr.fit(X_train, Y_train.astype('int'))
y_pred=regr.predict(X_val)
evaluate(y_pred,Y_val.astype('int'))
```

```
f1_score is: 0.243039
accuracy_score is: 8e-05
```

LightGBM

```
from lightgbm.sklearn import LGBMClassifier
from sklearn.multioutput import MultiOutputClassifier
lgbm = LGBMClassifier()           # create
lgbm_regr= MultiOutputClassifier(lgbm)
lgbm_regr.fit(X_train, Y_train.astype('int'))            # train
y_pred=lgbm_regr.predict(X_val)
evaluate(y_pred,Y_val.astype('int'))
```

```
f1_score is: 0.303339
accuracy_score is: 0.000199
```

3. Summary

The f1 score of the LightGBM is the highest which indicates that it might be the best predictor of descriptions of wine. However, there is a lot of possibility for optimization for the neural network model which may result in an improved performance.

Method	F1 score
Neural Network	0.29
GaussianNB	0.28
Perceptron	0.23
DecisionTree	0.25
SVC	0.17
SVC linear	0.19
K-nearest	0.23
Random-forest	0.23
Logistic-Regression	0.24
XGBoost	0.24
LightGBM	0.30

Fig 7.2.3.3 Summary of F1 scores in categorical models

7.3 Models comparison

7.3.1 Refinement

Neural network model has the potential to be refined both by adjusting hyperparameters but modifying the model itself. Both lightgbm and XGBoost also performed satisfactorily well although it takes considerable time to optimize and train the models, they do support GPU acceleration which will speed up the training significantly.

The first refinement in this project was done on Keras on number of hidden layer. A high number of hidden layers will cause the model to overfit while a low hidden layers will cause the model to not reach a good convergence.

8 Results

Summary of the performance result can be found in the table 8.1.1

Forecast parameter	Best model(s)	Performance indicator	Value	Benchmark value (linear regression)
Price	LGBM, Ridge, Neural Network	Mean absolute error, median absolute error (normalized data from 0-1)	0.005 ,0.003	

Wine Rating	LGBM, Ridge	Mean absolute error, median absolute error (normalized data from 0-1)	0.10,0.08	
Wine Description	LightGBM	f1	0.3034	0.2443 (logistic regression)

8.1 Model Evaluation and Validation

8.1.1 Price and Wine rating

It has been shown that the price and wine rating forecast are quite realistic and reliable according to the sample data shown in fig 8.1.1.

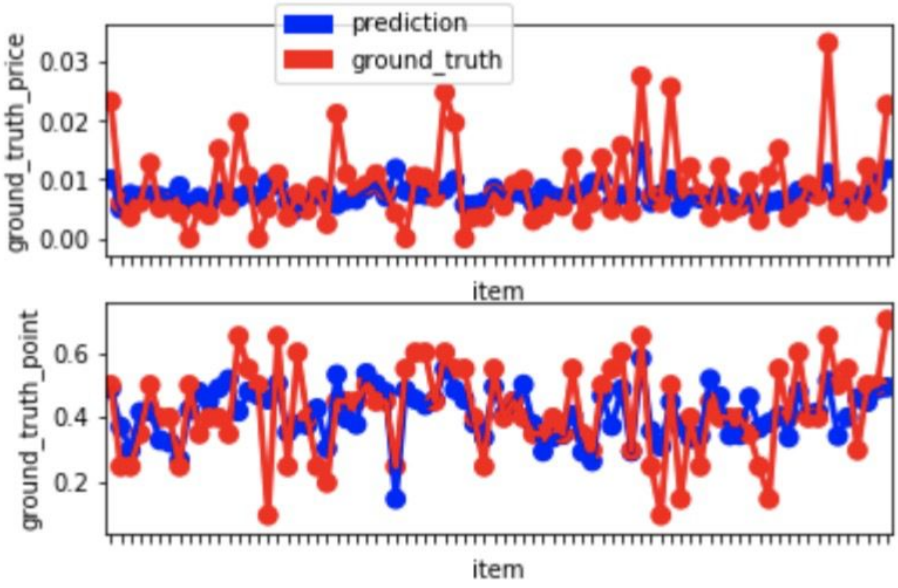


Fig 8.1.1 forecast of point and price compared to ground truth for datapoint 40-100

8.1.2 Wine Description forecast

As shown in fig 8.1.2 The prediction is still some way off the required value. A lot of optimization is still required for the model to produce a satisfactory description of wine.

	taste_aroma	taste_fruit	taste_herb	taste_palat	taste_offer	taste_appl	taste_citru	taste_acid	t
0	0	1	0	0	0	0	0	1	
1	0	1	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	
4	0	1	0	0	0	0	0	1	
5	0	0	0	0	0	0	0	0	
6	1	0	0	0	0	0	0	0	
7	0	0	0	1	0	0	0	0	
8	0	0	0	0	0	0	0	0	
9	1	0	0	1	0	0	0	0	
10	0	1	0	0	0	0	0	1	

Fig 8.1.2 prediction of a random 10 points by LightGBM model

taste_aroma	taste_fruit	taste_herb	taste_palat	taste_offer	taste_appl	taste_citru	taste_acid	taste_ripe	taste_fruiti
0	1	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	0	1
0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	1	1	0
0	0	0	0	0	0	0	1	1	0
1	0	0	0	1	1	1	0	0	0
1	0	0	0	1	0	0	1	0	0
0	0	1	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0
1	0	0	1	0	1	0	0	1	0
0	0	0	0	0	0	1	0	0	0

Fig 8.1.2.1 ground truth for the same random 10 points as fig 8.1.2

9 Conclusion

It has been shown that The wine data enhanced by weather is a very effective inputs to a forecast model of wine taste, price, and rating. If more details about the weather is included, and more training trials are used, the model could potentially be very accurate.

LGBM seems to be the current best model along with Ridge and Neural network. These models should be developed further together to find the best performing model.

10 Improvement

10.1 Additional tasks to launch the model

- Create a RESTApi with API gateway and aws lambda
- Create a webapp to interact with the gateway and gather input-output

10.2 Model improvement

The chosen neural network model can be improved by modifying various hyperparameter including number of nodes, layers, loss function, learning rate.

10.2.1 Further data enhancement

The weather data has been shown to have a strong relationship with both wine description, points, and price. However, only temperature has been included thus far. Other aspects of weather such as humidity, rainfall, wind speed, and forest fire rate are very likely to have a strong relation with the outputs and would greatly improve the model.

10.2.2 Additional training data resolution

The training data of temperature was a monthly data which gives less resolution than a daily data. There could be many underlying patterns to those which determines the quality, price and features of wine,

10.2.3 Optimization of hyperparameters

10.2.3.1 Optimization of number of hidden layer

It has been shown that the number hidden layers affects the convergence of model strongly. Models with a high number of hidden layer tend to converge earlier. However, they are prone to overfitting.

The original model has the neural network numbers [2440,2000,1000,500,2] which contains 3 hidden layers. The optimization aims to test the effects of incresing the hidden layer with 2000 width and see the effect on the accuracy of trained model while maintaining the same conditions in every other parameters.

Model node network [2440,2000*n,1000,500,2]

The result shows that the optimum number of n for the price prediction is 5 and for point is 1. It might be more efficient to create 2 different models for the 2 predictions.

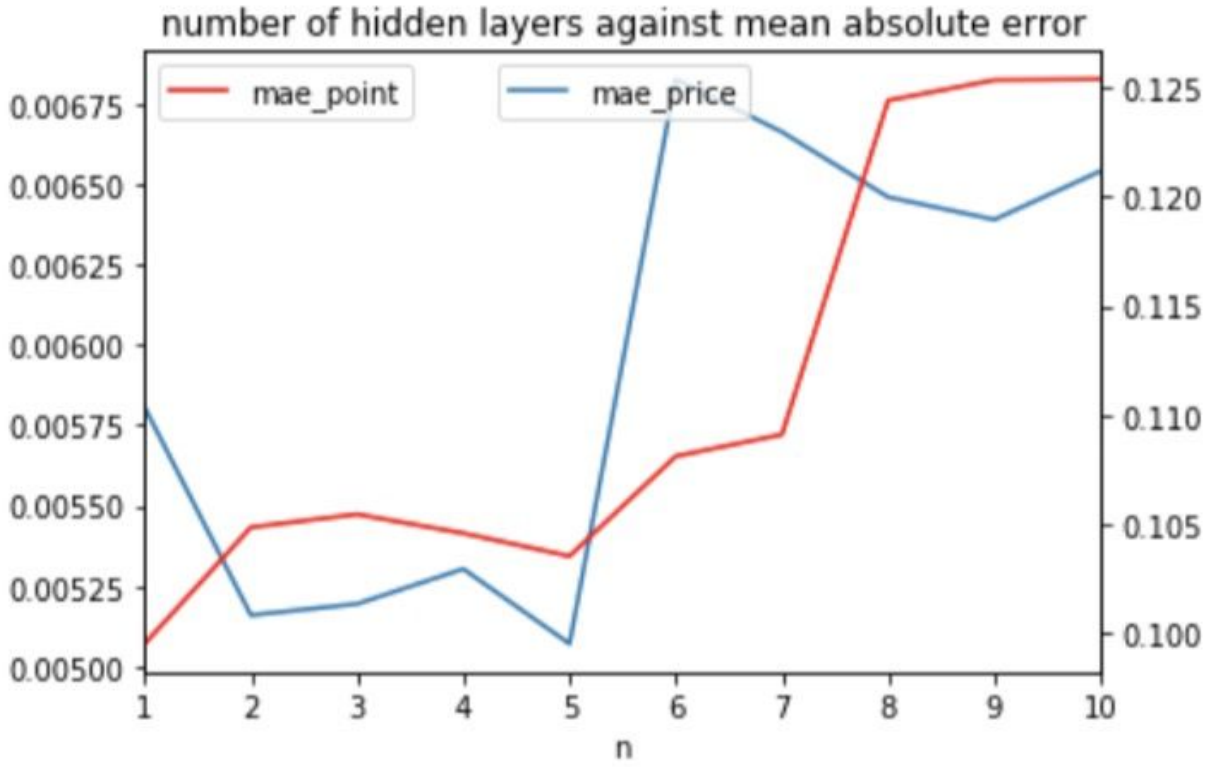


Fig 10.2.3.1 Plot of number of hidden layers [2440,2000*n,1000,500,2] n against mean average error in prediction for the neural network model.

Reference

1. Kaggle. 2019. Wine Reviews 130k wine reviews with variety, location, winery, price, and description. [ONLINE] Available at: https://www.kaggle.com/zynicide/wine-reviews. [Accessed 31 July 2019].

2. Winemag. 2019. Wine Enthusiast. [ONLINE] Available at: http://www.winemag.com/?s=&drink_type=wine. [Accessed 31 July 2019].
3. Kaggle. 2019. Climate Change: Earth Surface Temperature Data. [ONLINE] Available at: https://www.kaggle.com/berkeleyearth/climate-change-earth-surface-temperature-data. [Accessed 31 July 2019].
4. Robinson, S., 2019. Predicting the price of wine with the Keras Functional API and TensorFlow. Predicting the price of wine with the Keras Functional API and TensorFlow, [Online]. 1, 1. Available at: https://medium.com/tensorflow/predicting-the-price-of-wine-with-the-keras-functional-api-and-tensorflow-a95d1c2c1b03 [Accessed 31 July 2019].
5. Vonvino. 2019. Artificial Vintelligence: AI Gets Taste of Wine Industry. [ONLINE] Available at: https://vonvino.com/artificial-intelligence/. [Accessed 31 July 2019].
6. Freecodecamp. 2018. How to Use Data Science to Understand What Makes Wine Taste Good. [Online]. [31 July 2019]. Available from: https://www.freecodecamp.org/news/using-data-science-to-understand-what-makes-wine-taste-good-669b496c67ee/
7. Uciedu. 2019. Uciedu. [Online]. [31 July 2019]. Available from: https://archive.ics.uci.edu/ml/datasets/wine quality
8. cortez, P., 2019. Decision Support Systems. Modeling wine preferences by data mining from physicochemical properties, 47/4, 547-553.
9. Olivier goutay. 2018. Medium. [Online]. [31 July 2019]. Available from: https://towardsdatascience.com/wine-ratings-prediction-using-machine-learning-ce259832b321
10. https://geopy.readthedocs.io/en/stable/
11. https://climateknowledgeportal.worldbank.org/download-data
12. https://nominatim.org/release-docs/develop/api/Overview/
13. https://en.wikipedia.org/wiki/List_of_Appellation_d%27Origine_Contr%C3%B4l%C3%A9e_wines
14. https://github.com/seatgeek/fuzzywuzzy
15. https://www.nltk.org/
16. https://keras.io/getting-started/sequential-model-guide/
17. https://lightgbm.readthedocs.io/en/latest/Features.html
18. https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf
19. https://ncss-wpengine.netdna-ssl.com/wp-content/themes/ncss/pdf/Procedures/NCSS/Ridge_Regression.pdf