HOUSING PRICE PREDICTION - MACHINE LEARNING IN PYTHON

NCG 613 [A] - DATA ANALYTICS PROJECT ASSIGNMENT 1

Technical Report

RAHUL PAGARIA
STUDENT NUMBER 17251662

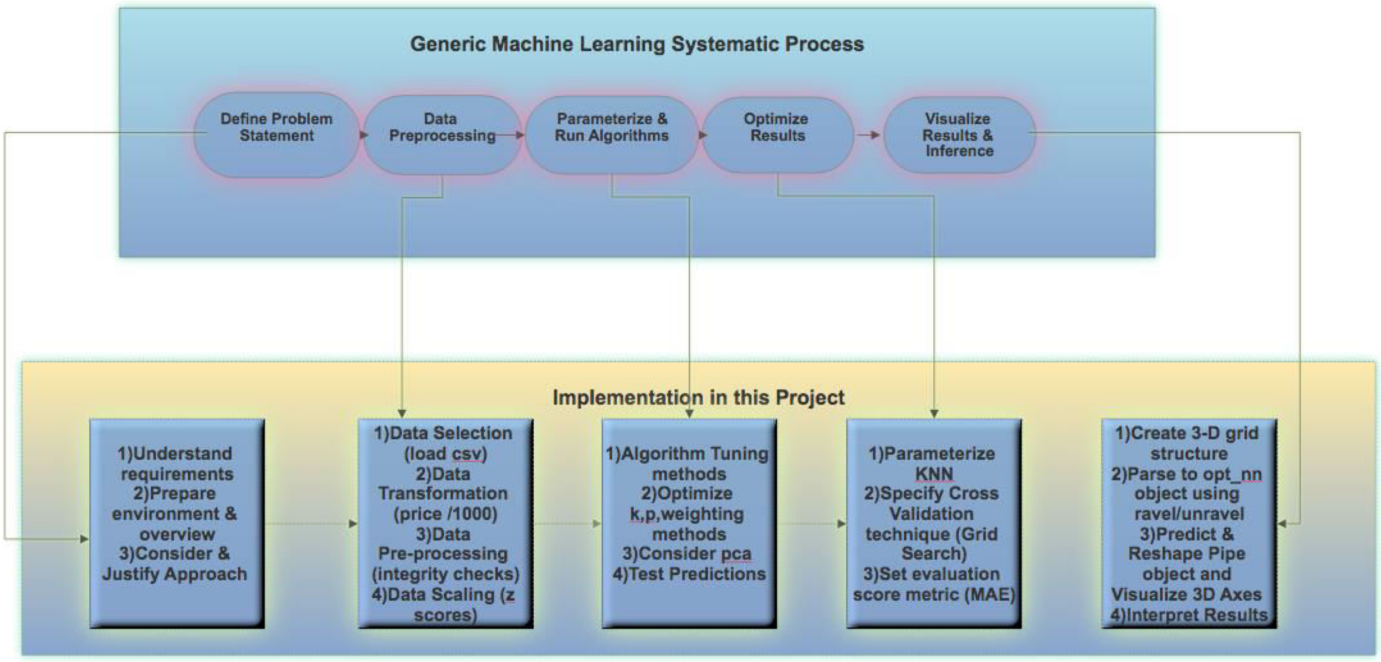
TABLE OF CONTENTS

Abstract

This project aims to construct a house price model in Python from the given dataset of London house prices. Implementation is done using the k nearest neighbors algorithm, for which a series of steps in the machine learning systematic process cycle are followed and justified. Optimal model selection and parameter tuning is done using score metrics via grid search cross validation and summarized.

Upon realization, we aim to visualize the said model with customizable accuracy to plot house prices in 3-D for a given set of floor areas across the co-ordinates of London.

Workflow and Understanding


Generic Machine Learning Systematic Process
Define Problem Statement
Data Preprocessing
Parameterize & Run Algorithms
Optimize Results
Visualize Results & Inference
Implementation in this Project
1)Understand requirements 2)Prepare environment & overview 3)Consider & Justify Approach
1)Data Selection (load csv) 2)Data Transformation (price /1000) 3)Data Pre-processing (integrity checks) 4)Data Scaling (z scores)
1)Algorithm Tuning methods 2)Optimize k,p,weighting methods 3)Consider pca 4)Test Predictions
1)Parameterize KNN 2)Specify Cross Validation technique (Grid Search) 3)Set evaluation score metric (MAE)
1)Create 3-D grid structure 2)Parse to opt_nn object using ravel/unravel 3)Predict & Reshape Pipe object and Visualize 3D Axes 4)Interpret Results

1. Data Preparation and Pre-processing

The ML procedure in this particular case is "supervised learning", i.e. since we have the response output in our dataset which will act as our training data model.

The dataset consists of floor area in sq.m, x & y co-ordinates (3 predictors) and price in pounds (response). Data is checked for nulls and basic statistics are generated for the response "price" and no explicit data cleaning was required.

1.1 Normalizing Data (Scaling)

Since the predictors are in different units of measure and encompass a wide range of values, it is important to standardize data. This is a variance stabilization exercise. Relative influence of one predictor changes greatly depending on the unit used. To remove this bias, data is rescaled using one of the many methods prior to algorithm run.

The function used is StandardScaler from the sklearn package. Z- Score is the method of re-scaling the data using the formula z = mean/std dev, this gives a score independent of the unit of measure. Here we use the z score approach.The fit method calibrates the object with default parameters and transform method stores the predictors in a rescaled column wise array.

2. KNN Machine Learning Algorithm Selection & Understanding Parameters

2.1 Justification

The k nearest neighbor algorithm mirrors the assessment technique of house price prediction in the real world, and with the parameters being aligned to the assessment done, the choice of the this algorithm is justified. KNN Regression is not necessarily the simplistic answer to future prediction of events, but it is valid in the local sense for a range bound data prediction like ours.

The parameters are considered for optimization are :-

1. The k (number of neighbors to consider to correspond to lowest error rate and optimize performance as well as avoid overfitting),
2. The weighting function (The default value, weights is 'uniform', assigns uniform weights to each neighbor. weights = 'distance' assigns weights proportional to the inverse of the distance from the query point [1]) and
3. The distance metric (Example : One of many metrics like Chebychev distance, Mahalanobis distance, Hamming distance and Cosine similarity.
   In our case, either Euclidean distance or city block distance are relevant)

2.2 Tuning Parameters

Evaluation of the model scoring has to be specified, using the sklearn metrics package wherein the function make_scorer creates the object to be used. The model is evaluated based on this parameter, and the value of mean absolute error is chosen.

Measure of accuracy is given by M.A.E = average of the absolute errors |e{i}|=|y{i}-x{i}|} where yi is the prediction and xi the true value.
Smaller value of MAE implies better performance.

For tuning parameters, looping through each of the possible combinations of k, p and the weighting method used in the nearest- neighbor averages, and applying the cross validation procedure for each combination of values. This is referred to as a *cross-validation grid search*.

This is done using a function in sklearn.grid search called *GridSearchCV*. The GridSearchCV function takes a number of arguments. Firstly estimator specifies the machine learning algorithm which is NN. Default values of cv are used, implying 50% of the data to be the test set, and the remaining 50% to be the training set. Further to this, the test and train set are swapped, and the results are computed by an average of both iterations.

Each element in the GridSearchCV function is called as a key-value pair, with the key being the parameter name and the argument is the parameter value setting. For example, param_grid is a dictionary containing the key value pairs as below = *{'n_neighbors':range(1,35),'weights':['uniform','distance'],'p':[1,2]})*

p is defined as the p value in the *Minkowski Metric*. The value from the function returned and stored in opt nn is similar to the regression object, and has fit and predict methods. Finally, an attribute of opt nn called best estimator provides the regression model that performed best - and this has a method called get params that returns a dictionary containing the tuning parameters.

3. Implementation via sklearn Pipeline

Pipelines concept involves the linear sequence of blocks to be chained together resulting in modeling that can be validated and instantiated. Herewith, all of the steps in the pipeline are stitched together are constrained to the data available for the evaluation, such as the training dataset or each fold of the cross validation procedure.The dictionary part of the param grid argument is <name>__<parameter> where <name> is the block name in the pipeline, and <parameter> is the tuning parameter name.

Here our code is:

```
pipe = Pipeline([('zscores',StandardScaler()),('NNreg',NN())])
opt_nn2 = GridSearchCV(
                estimator = pipe,
                scoring = mae,param_grid = {
                        'NNreg__n_neighbors':range(1,35), # k range from 1 to 35
                        'NNreg__weights':['uniform','distance'], # both functions
                        'NNreg__p':[1,2] # both Euclidean & city block metrics
                            }
               )
```

4. Visualizations

Function surf3d

Input parameters : Object pipe with parameterized GridSearchCV instance, floor area , color map scheme

Return : matplotlib.figure.Figure

We would like to present the data in the form of a surface 3D plot with the z axis being the predicted price of the house for a given standard floor area and location (x coordinate : Easting & y co-ordinate Northing), The structural grid of 100 * 100 (x,y) is created and normalized for the range of Easting & Northings.

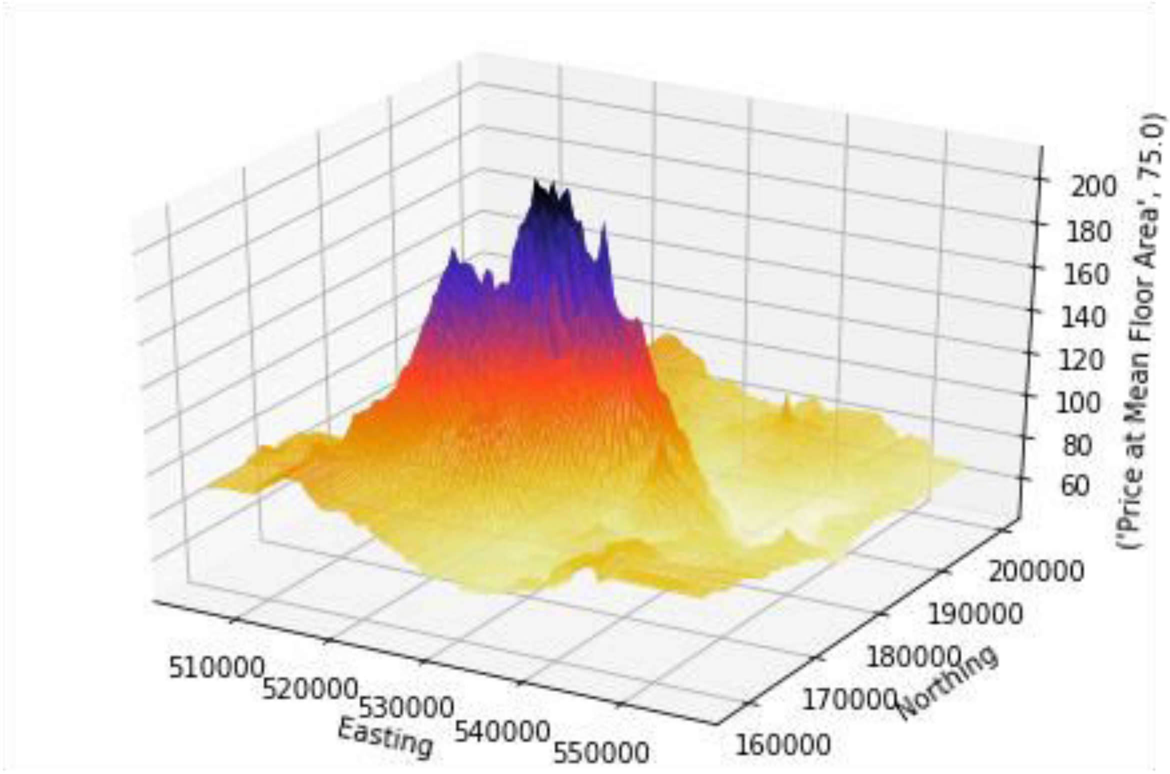
Axes 3D takes data values as 2-D arrays, and these arrays are parsed to simple arrays using ravel() to pass input predictor values in the form of matrix of { easting,northing, fl_area }. Once the house price response is collected , it is again parsed into a 2-D array and passed in the plot_surface as the z axis. The figure is labeled and row & columns step sizes are set to 1.

Following are the visualizations

4.1 Output Graphs

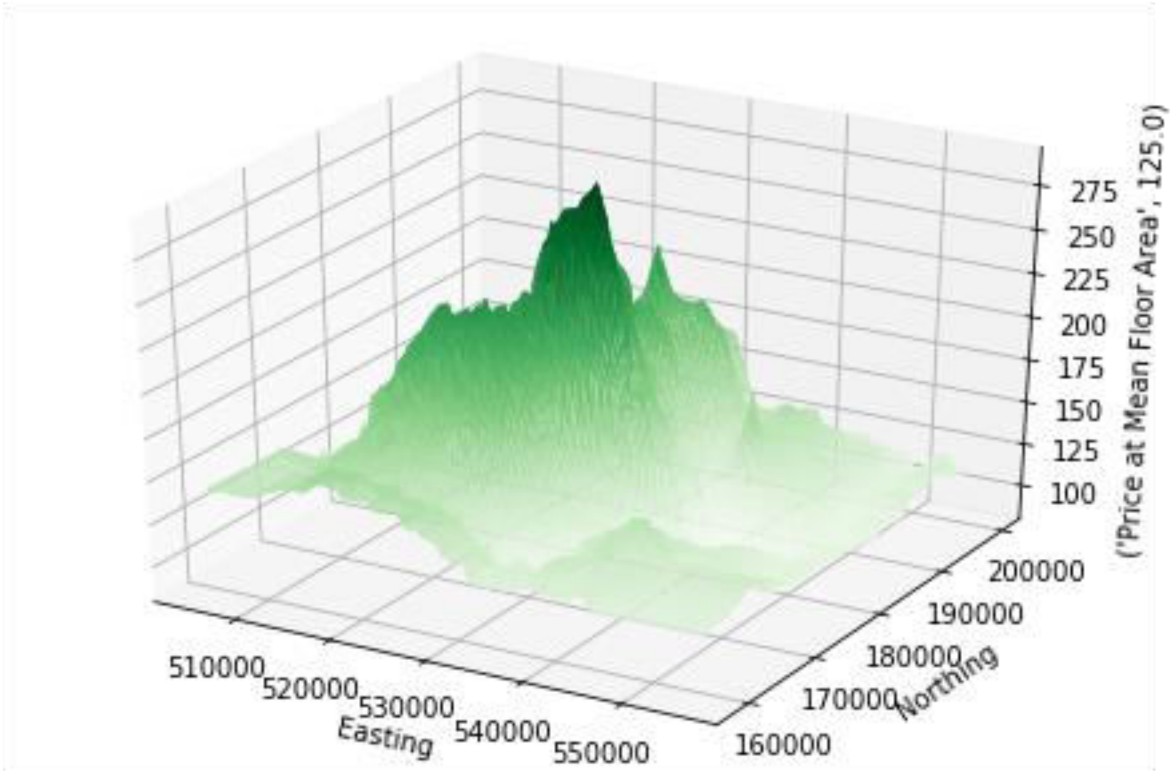
A 3-D plot of the house price for floor areas of 75 square metres

```
surf3d(opt_nn2, 75.0,"CMRmap_r")
pl.show()
```



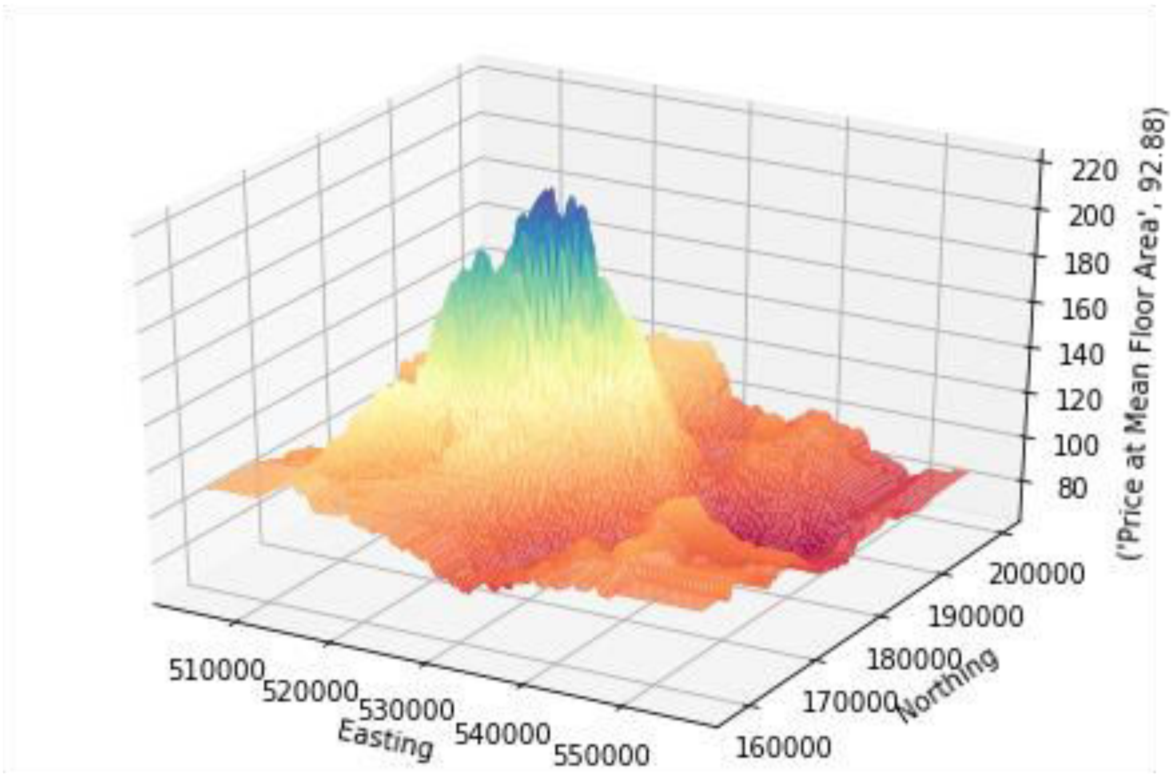
<u>A 3-D plot of the house price for floor areas of 125 square meters</u>

```
surf3d(opt_nn2, 125.0,"Greens")
pl.show()
```



A 3-D plot of the house price for average floor area

```
surf3d(opt_nn2, round(np.mean(hp.fl_area),2),"Spectral")
pl.show()
```



In [75]:

5 Summary and Conclusion

Final tuning parameters chosen by cross-validation can be retreived - and also its MAE score .

An attribute of opt nn called best-estimator- provides the regression model that performed best - and this has a method called get params that returns a dictionary containing the tuning parameters

Example : print (opt_nn2.best_estimator_.get_params())

```
{'memory': None, 'steps': [('zscores', StandardScaler(copy=True, with_mean=True, with_std=True)),
('NNreg', KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=13, p=1,
          weights='distance'))], 'zscores': StandardScaler(copy=True, with_mean=True, with_std=True),
'NNreg': KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=13, p=1,
          weights='distance'), 'zscores__copy': True, 'zscores__with_mean': True, 'zscores__with_std':
True, 'NNreg__algorithm': 'auto', 'NNreg__leaf_size': 30, 'NNreg__metric': 'minkowski',
'NNreg__metric_params': None, 'NNreg__n_jobs': 1, 'NNreg__n_neighbors': 13, 'NNreg__p': 1,
'NNreg__weights': 'distance'}
```

Our neatly summarized output being the following :--

Summary using pipeline

Attribute Key	***Parameter Value***
Nearest neighbours	**13**
Minkowski p	**1**
Weighting	**distance**
MAE Score	**26.47**

The optimal parameters are: use 13 nearest neighbors, based on a city block metric (p = 1) and a distance weighted mean as a predictor. This achieves an mean absolute error of around 26.7 thousand pounds

A principal components analysis (PCA) stage can been added to the pipeline but that it has an identical score to the model without any PCA component. On verifying the summary stats, we conclude that no advantage is acheived here by reducing to 1 or 2 components.

6 Appendix

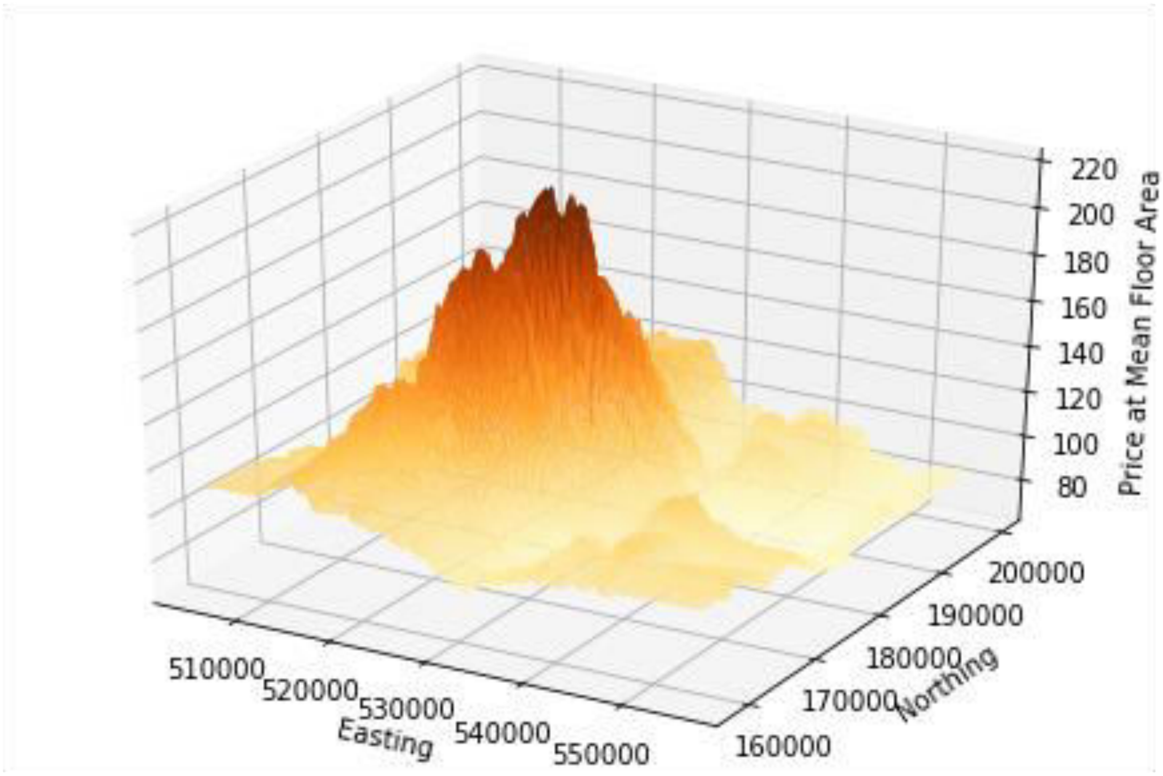
6.1 Project Github URL

https://github.com/rahulpagaria/Housing-Price-Python

6.2 Code

```
import numpy as np
import pandas as pd
import os
import pylab as pl
from mpl_toolkits.mplot3d import Axes3D
In [61]:
os.chdir("/Users/rahulpagaria/Downloads/")

hp = pd.read_csv('hpdemo.csv', dtype=float) # creating dataframe containing 1405
rows of house price data of London in 1990

print (hp.isnull().values.any())

import matplotlib.pyplot as plt

#%matplotlib inline
#plt.boxplot(hp.fl_area)

False
In [62]:
minimum_price = np.min(hp.price)
maximum_price = np.max(hp.price)
mean_price = np.mean(hp.price)
median_price = np.median(hp.price)
std_price = np.std(hp.price)
first_quartile = np.percentile(hp.price, 25)
third_quartile = np.percentile(hp.price, 75)
inter_quartile = third_quartile - first_quartile

# Show the calculated statistics
print ("Statistics for London housing dataset:\n")
print ("Minimum price: ${:,.2f}".format(minimum_price))
print ("Maximum price: ${:,.2f}".format(maximum_price))
print ("Mean price: ${:,.2f}".format(mean_price))
print ("Median price ${:,.2f}".format(median_price))
print ("Standard deviation of prices: ${:,.2f}".format(std_price))
print ("First quartile of prices: ${:,.2f}".format(first_quartile))
print ("Second quartile of prices: ${:,.2f}".format(third_quartile))
print ("Interquartile (IQR) of prices: ${:,.2f}".format(inter_quartile))

Statistics for London housing dataset:

Minimum price: $32,500.00
Maximum price: $850,000.00
Mean price: $120,668.33
Median price $100,000.00
Standard deviation of prices: $76,519.85
First quartile of prices: $73,750.00
Second quartile of prices: $140,000.00
Interquartile (IQR) of prices: $66,250.00
In [63]:
from sklearn.preprocessing import StandardScaler
x_scaler = StandardScaler()
x_scaler.fit(hp[['east','north','fl_area']])
X = x_scaler.transform(hp[['east','north','fl_area']])
print(X[:5,:]) # to print first 5 rows and all columns (easting, northing and fl_Area)

[[-0.46109525 -0.0036912  -1.16501944]
 [ 0.39051366 -1.02696462 -0.73029338]
 [-1.29458482 -0.45718737 -0.07820428]
 [-1.16774945 -1.01533651  0.87275899]
 [ 0.435812   -1.22464244 -1.16501944]]
In [64]:
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsRegressor as NN

pipe = Pipeline([('zscores', StandardScaler()),('NNreg', NN(n_neighbors=6,
weights='uniform', p=2))])
In [65]:
print (pipe)
price = hp['price']/1000.00
pipe.fit(hp[['east','north','fl_area']],price)
print(pipe.predict([[523800.0,179750.0,55.0]]))

Pipeline(memory=None,
     steps=[('zscores', StandardScaler(copy=True, with_mean=True,
with_std=True)), ('NNreg', KNeighborsRegressor(algorithm='auto', leaf_size=30,
metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=6, p=2,
          weights='uniform'))])
[128.5]
In [66]:
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, make_scorer
mae = make_scorer(mean_absolute_error, greater_is_better=False)

pipe = Pipeline([('zscores',StandardScaler()),('NNreg',NN())])
opt_nn2 = GridSearchCV(
                    estimator = pipe,
                    scoring = mae,param_grid = {
                                        'NNreg__n_neighbors':range(1,35),
                                    'NNreg__weights':['uniform','distance'],
                                    'NNreg__p':[1,2]
                                    }
                )
opt_nn2.fit(hp[['east','north','fl_area']],price)
print(opt_nn2.predict([[523800.0, 179750.0, 55.0]]))

[121.78783506]
In [67]:
east_mesh, north_mesh = np.meshgrid(np.linspace(505000, 555800,
100),np.linspace(158400, 199900, 100))
fl_mesh = np.zeros_like(east_mesh)
fl_mesh[:,:] = np.mean(hp['fl_area'])
print(east_mesh.shape)
print(north_mesh.shape)

(100, 100)
(100, 100)
In [68]:
grid_predictor_vars = np.array([east_mesh.ravel(),north_mesh.ravel(),
fl_mesh.ravel()]).T
hp_pred = opt_nn2.predict(grid_predictor_vars)
hp_mesh = hp_pred.reshape(east_mesh.shape)
In [69]:
def print_summary2(opt_pipe_object):
    params = opt_pipe_object.best_estimator_.get_params()
    score = - opt_pipe_object.best_score_
    print ("Nearest neighbours: %8d" % params['NNreg__n_neighbors'])
    print ("Minkowski p : %8d" % params['NNreg__p'])
    print ("Weighting : %8s" % params['NNreg__weights'])
    print ("MAE Score : %8.2f" % score)
    return

print("Summary using pipeline:")
print_summary2(opt_nn2)

#print (hp[])
print(hp.loc[hp['east'] == 523800.0])
print(opt_nn2.predict([[523800.0, 179700.0, 40.0]]))

Summary using pipeline:
Nearest neighbours:       13
Minkowski p :        1
Weighting : distance
MAE Score :    26.47
        ID      east     north     price  fl_area
0      1.0  523800.0  179700.0  107000.0     50.0
22    23.0  523800.0  184600.0  285000.0    198.0
150  151.0  523800.0  175000.0  100000.0     45.0
158  159.0  523800.0  169900.0  132500.0     77.0
[98.55535845]
In [70]:
fig = pl.figure()
ax = Axes3D(fig)
# Plot the surface.# Add a color bar which maps values to colors.
ax.plot_surface(east_mesh, north_mesh, hp_mesh, rstride=1,cstride=1,
cmap='YlOrBr', lw=0.01)
ax.set_xlabel('Easting')
ax.set_ylabel('Northing')
ax.set_zlabel('Price at Mean Floor Area')
```

```
pl.show()
```



```
In [71]:
# Function surf3d to plot the predicted values
def surf3d(pipe_model,fl_area,cscheme):  # Input args pipe object, fl_area integer & color scheme string
    east_mesh, north_mesh = np.meshgrid(
    np.linspace(505000,555800,100),
    np.linspace(158400,199900,100)) # Create a structural background grid
    fl_mesh = np.zeros_like(east_mesh)
    fl_mesh[:,:] = fl_area
    grid_predictor_vars = np.array([east_mesh.ravel(),
    north_mesh.ravel(),fl_mesh.ravel()]).T        # Convert to single array using ravel & transpose
    hp_pred = pipe_model.predict(grid_predictor_vars)
    hp_mesh = hp_pred.reshape(east_mesh.shape)
    fig = pl.figure()
    ax = Axes3D(fig)
    ax.plot_surface(east_mesh, north_mesh, hp_mesh, # Axes define
    rstride=1, cstride=1, cmap=cscheme,lw=0.01)
    ax.set_xlabel('Easting')
    ax.set_ylabel('Northing')
    zl = 'Price at Mean Floor Area',fl_area
    ax.set_zlabel(zl)
    return
In [72]:

from sklearn.decomposition import PCA

pipe1 = Pipeline([('zscores',StandardScaler()),('prcomp',PCA()),('NNreg',NN())])

opt_nn3 = GridSearchCV(
                    estimator = pipe1,
                        scoring = mae,
                        param_grid = {
                                'NNreg__n_neighbors':range(1,35),
                                'NNreg__weights':['uniform','distance'],
                                'NNreg__p':[1,2],
                                'prcomp__n_components':[1,2,3]
                                }
                        )

opt_nn3.fit(hp[['east','north','fl_area']],price)

print(opt_nn3.best_estimator_.get_params()['prcomp__n_components'])
print(opt_nn3.best_score_)

3
-26.603830189496502
```

7 References

1. http://scikit-learn.org/stable/modules/neighbors.html
2. https://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html
3. http://mpastell.com/2013/05/02/matplotlib_colormaps/
4. https://machinelearningmastery.com/machine-learning-checklist/