kaggle1

October 19, 2025

1 0.99318 Is the final score I got

[11]:
```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler,␣
 ↪OneHotEncoder,PolynomialFeatures #preprocessing
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.pipeline import Pipeline #pipeline stuff
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.linear_model import LinearRegression #classifer
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

#importing the various documents
train_data=pd.read_csv("train.csv")
test_data=pd.read_csv("test.csv")
X_train = train_data.drop(columns=['SalePrice'])
y_train = train_data["SalePrice"]
X_test = test_data #no labels vec to confirm results,
```

[12]:
```python
numeric_features =␣
 ↪['LotFrontage','LotArea','OverallQual','OverallCond','YearBuilt','YearRemodAdd',
                  ␣
 ↪'MasVnrArea','BsmtFinSF1','BsmtFinSF2','BsmtUnfSF','TotalBsmtSF','1stFlrSF','2ndFlrSF',
                  ␣
 ↪'LowQualFinSF','GrLivArea','BsmtFullBath','BsmtHalfBath','FullBath','BedroomAbvGr','Kitchen
                  ␣
 ↪'TotRmsAbvGrd','Fireplaces','GarageYrBlt','GarageCars','GarageArea','WoodDeckSF','OpenPorch
                  ␣
 ↪'EnclosedPorch','3SsnPorch','ScreenPorch','PoolArea','MiscVal','MoSold','YrSold']

ordinal_features = ['MSSubClass',␣
 ↪'LotShape','LandContour','LandSlope','BldgType','HouseStyle','ExterQual'
```

```
                   ␣
↪,'ExterCond','BsmtQual','BsmtCond','BsmtExposure','BsmtFinType1','BsmtFinType2',
                   ␣
↪'HeatingQC','CentralAir','KitchenQual','Functional','FireplaceQu','GarageFinish',
                    'GarageQual','GarageCond','PavedDrive','PoolQC','Fence']

oneHotEncode_features =␣
↪['Alley','LotConfig','Neighborhood','Condition1','Condition2','RoofStyle',
                        ␣
↪'RoofMatl','Exterior1st','Exterior2nd','MasVnrType','Foundation','Heating',
                        ␣
↪'Electrical','GarageType','MiscFeature','SaleType','SaleCondition']
```

This is the selection of features that I have choosen to use. This is only a subset of the total number of features, but I attempted to select features that seemed the most relavent for predicting house prices while not selecting too many and cause some overfitting to the data, but also enough so that the complex aspects of the data can still be captured.

[13]:
```
numerical_x_train = X_train[numeric_features]
numerical_x_train.isna().sum()
```

[13]:
```
LotFrontage      259
LotArea            0
OverallQual        0
OverallCond        0
YearBuilt          0
YearRemodAdd       0
MasVnrArea         8
BsmtFinSF1         0
BsmtFinSF2         0
BsmtUnfSF          0
TotalBsmtSF        0
1stFlrSF           0
2ndFlrSF           0
LowQualFinSF       0
GrLivArea          0
BsmtFullBath       0
BsmtHalfBath       0
FullBath           0
BedroomAbvGr       0
KitchenAbvGr       0
TotRmsAbvGrd       0
Fireplaces         0
GarageYrBlt       81
GarageCars         0
GarageArea         0
WoodDeckSF         0
```

```
OpenPorchSF      0
EnclosedPorch    0
3SsnPorch        0
ScreenPorch      0
PoolArea         0
MiscVal          0
MoSold           0
YrSold           0
dtype: int64
```

Now as I plan to encode the catagorical features in this model I need to check whether there are any missing numerical features that will need to be filled in. This is what i have done above. As can be seen by the fact there are several found NaN values, I am going to have to impute these values using a imputer.

[14]:
```
#preproccessors
numeric_processor=Pipeline(steps = [
    ('imputer',SimpleImputer(strategy='mean')),
    ('scaler',StandardScaler())
    ])
oneHotencoder_processor=Pipeline(steps = [
    ('imputer',SimpleImputer(strategy='constant')),
    ('1Hotencoder',OneHotEncoder(handle_unknown='ignore',min_frequency=.01))
    ])
ordinalencoder_processor = Pipeline(steps = [
    ('imputer',SimpleImputer(strategy='constant')),
   ␣
  ↪('ordinalencoder',OrdinalEncoder(handle_unknown='use_encoded_value',unknown_value=-1,min_fr
  ↪01))
])
```

Along with encoders to encode the catagorical features i am going to ensure that they handle unknowns on there own by inputing their own values along with ignoring any labels that barely have any occurances

[15]:
```
#transfromers
feature_processor = ColumnTransformer(transformers=[
    ('num',numeric_processor,numeric_features),
    ('1hotEncode',oneHotencoder_processor,oneHotEncode_features),
    ('ordinalencoder',ordinalencoder_processor,ordinal_features)],
    remainder='drop'
    )
```

[16]:
```
#final pipeline
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import Ridge
pipe_clf = Pipeline(steps = [
    ('feature_process',feature_processor),
    ('poly_features',PolynomialFeatures()),
    ('elsatic_clf',Ridge())
])
pipe_clf
```

[16]:
```
Pipeline(steps=[('feature_process',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer()),
                                                                  ('scaler',
StandardScaler())]),
                                                  ['LotFrontage', 'LotArea',
                                                   'OverallQual', 'OverallCond',
                                                   'YearBuilt', 'YearRemodAdd',
                                                   'MasVnrArea', 'BsmtFinSF1',
                                                   'BsmtFinSF2', 'BsmtUnfSF',
                                                   'TotalBsmtSF', '1stFlrSF',
                                                   '2ndFlrSF', 'LowQualFinSF',
                                                   'GrLivArea', 'BsmtFul…
                                                  ['MSSubClass', 'LotShape',
                                                   'LandContour', 'LandSlope',
                                                   'BldgType', 'HouseStyle',
                                                   'ExterQual', 'ExterCond',
                                                   'BsmtQual', 'BsmtCond',
                                                   'BsmtExposure',
                                                   'BsmtFinType1',
                                                   'BsmtFinType2', 'HeatingQC',
                                                   'CentralAir', 'KitchenQual',
                                                   'Functional', 'FireplaceQu',
                                                   'GarageFinish', 'GarageQual',
                                                   'GarageCond', 'PavedDrive',
                                                   'PoolQC', 'Fence'])])),
                ('poly_features', PolynomialFeatures()),
                ('elsatic_clf', Ridge())])
```

Now we have created the total pipeline of encoders and processors and are now ready to train the model. Additionally we are using Elsatic_Net as our linear regression model so that we can use its regularization to effectively downplay potentially non particapating features. On top of this we are using polynomial features to try and capture more complexities in the data.

Because we are using elsatic net we are going to need to try and find the ideal alpha value for the model to implament along with a ideal polynomial degree for the polynomial features, which is what I do below with a grid search.

[17]:
```
param_dict = {'elsatic_clf__alpha':np.linspace(0,1,20),'poly_features__degree':
 ↪[1,2]}
grid = GridSearchCV(pipe_clf,
                    param_dict,
                    cv=5,
                    scoring='accuracy',
                    verbose=1,
                    n_jobs=-1
                    )
```

[18]:
```
import warnings
warnings.simplefilter(action='ignore') #some warnings appear because the model␣
 ↪gets confused thinking this is trying to be a catagorical model
pipe_clf.fit(X_train,y_train)
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 40 candidates, totalling 200 fits
```

[18]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('feature_process',
                                        ColumnTransformer(transformers=[('num',
Pipeline(steps=[('imputer',
          SimpleImputer()),
         ('scaler',
          StandardScaler())]),
['LotFrontage',
'LotArea',
'OverallQual',
'OverallCond',
'YearBuilt',
'YearRemodAdd',
'MasVnrArea',
'BsmtFinSF1',
'BsmtFinSF2',
'BsmtUnfSF',
'TotalBsmtSF',
'1stFlrSF',
'2ndFlrSF',
'LowQua…
                                       ('poly_features', PolynomialFeatures()),
                                       ('elsatic_clf', Ridge())]),
             n_jobs=-1,
             param_grid={'elsatic_clf__alpha': array([0.        , 0.05263158,
0.10526316, 0.15789474, 0.21052632,
       0.26315789, 0.31578947, 0.36842105, 0.42105263, 0.47368421,
       0.52631579, 0.57894737, 0.63157895, 0.68421053, 0.73684211,
       0.78947368, 0.84210526, 0.89473684, 0.94736842, 1.        ]),
                         'poly_features__degree': [1, 2]},
             scoring='accuracy', verbose=1)
```

[19]:
```
print(grid.best_estimator_)
```

```
Pipeline(steps=[('feature_process',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('imputer',
SimpleImputer()),
                                                                  ('scaler',
StandardScaler())]),
                                                  ['LotFrontage', 'LotArea',
                                                   'OverallQual', 'OverallCond',
                                                   'YearBuilt', 'YearRemodAdd',
                                                   'MasVnrArea', 'BsmtFinSF1',
                                                   'BsmtFinSF2', 'BsmtUnfSF',
                                                   'TotalBsmtSF', '1stFlrSF',
                                                   '2ndFlrSF', 'LowQualFinSF',
                                                   'GrLivArea', 'BsmtFul…
                                                   'BldgType', 'HouseStyle',
                                                   'ExterQual', 'ExterCond',
                                                   'BsmtQual', 'BsmtCond',
                                                   'BsmtExposure',
                                                   'BsmtFinType1',
                                                   'BsmtFinType2', 'HeatingQC',
                                                   'CentralAir', 'KitchenQual',
                                                   'Functional', 'FireplaceQu',
                                                   'GarageFinish', 'GarageQual',
                                                   'GarageCond', 'PavedDrive',
                                                   'PoolQC', 'Fence'])])),
                ('poly_features', PolynomialFeatures(degree=1)),
                ('elsatic_clf', Ridge(alpha=np.float64(0.0)))])
```

This then is our best model, interestingly enough the gridsearch found best performance with polynomial features of degree 1, and a elsatic net alpha value of 0, meaning it found best performance out of no regularization and no polynomial features. This most likely is beacuse of the large size of the dataset and the large number of features we are trying to use, regardless this is the model we have.

So now below you can see the we are doing the final steps for the submission to Kaggle, getting the output, reshaping to column vectors so we can concatenate the output to an index, turning to a dataframe and then outing to a csv, and that is all.

[20]:
```python
pred = grid.predict(X_test).reshape(-1,1)
index_arr = np.arange(1461, 1461+pred.size).reshape(-1,1)
index_arr = index_arr.astype(int)
final_pred = np.concatenate((index_arr, pred),axis=-1)
df = pd.DataFrame(final_pred,columns=['Id','SalePrice'])
df['Id'] = df['Id'].astype(int)
df.to_csv('output.csv', index=False)
```

Here is the score I got: 0.99318 Which is pretty bad and sad but oh well! Theres the model, overall I think this project went well, it helped me learn how to successfully manage a large dataset and make a model out of it, even if the model didnt perform the best.

```
[ ]:
```