pds2136_Task1

February 19, 2020

[15]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_openml

from sklearn.model_selection import␣
 ↪train_test_split,cross_val_score,GridSearchCV,KFold
from sklearn.preprocessing import OneHotEncoder,StandardScaler
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline, Pipeline

from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import LinearSVC
```

[3]:
```
data = fetch_openml("credit-g")
```

[4]:
```
X = data["data"]
y = data["target"]
features = data["feature_names"]
```

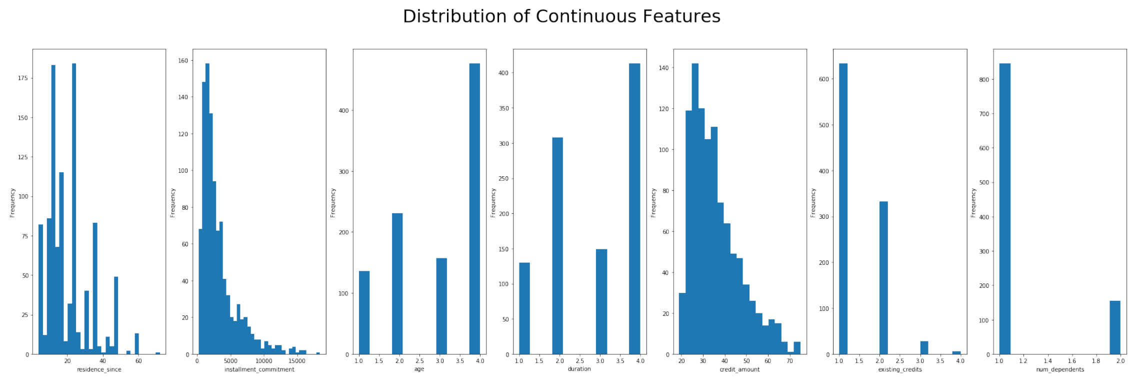
[5]:
```
categorical_columns = list(data["categories"].keys())
continuous_columns = list(set(features) - set(categorical_columns))
category_values = list(data["categories"].values())
```

[6]:
```
categorical_columns_index = [i for i,v in enumerate(features) if(v in␣
 ↪categorical_columns)]
continuous_columns_index = [i for i,v in enumerate(features) if(v in␣
 ↪continuous_columns)]
```

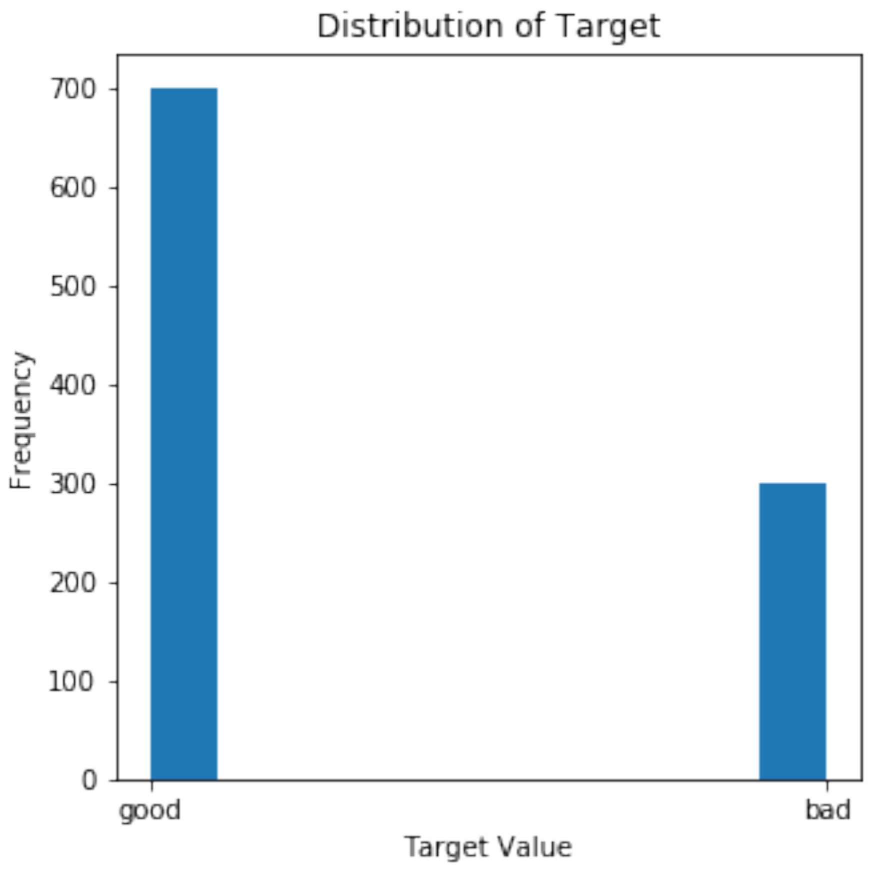
[7]:
```
print("- Following is the list of continuous features:")
print(continuous_columns)
print()
print("- Following is the list of categorical features:")
print(categorical_columns)
```

```
- Following is the list of continuous features:
['residence_since', 'installment_commitment', 'age', 'duration',
'credit_amount', 'existing_credits', 'num_dependents']

- Following is the list of categorical features:
['checking_status', 'credit_history', 'purpose', 'savings_status', 'employment',
'personal_status', 'other_parties', 'property_magnitude', 'other_payment_plans',
'housing', 'job', 'own_telephone', 'foreign_worker']
```

[8]:
```
fig, axes = plt.subplots(1, len(continuous_columns_index) , figsize=(35, 10))

for i in range(len(continuous_columns_index)):
    axes[i].hist(X[:,continuous_columns_index[i]],bins = "auto")
    axes[i].set_xlabel(continuous_columns[i])
    axes[i].set_ylabel("Frequency")

fig.suptitle("Distribution of Continuous Features", fontsize=32)
plt.show()
```



[9]:
```
fig, axes = plt.subplots(figsize=(5, 5))
axes.hist(y)
axes.set_xlabel("Target Value")
axes.set_ylabel("Frequency")
axes.set_title("Distribution of Target")

plt.show()
```



[10]:
```
X_df = pd.DataFrame(X, columns = features)
y_df = pd.DataFrame(y, columns = ["target"])
```

[11]:
```
X_train_val, X_test, y_train_val, y_test = train_test_split(X_df, y_df,␣
 ↪test_size=0.20, random_state=42)
```

[12]:
```
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val,␣
 ↪test_size=0.20, random_state=42)
```

[16]:
```
ohe = OneHotEncoder(categorical_features=categorical_columns_index,␣
 ↪handle_unknown="ignore")
ohe = ohe.fit(X_train)
X_train_preprocessed = ohe.transform(X_train)
X_val_preprocessed = ohe.transform(X_val)
```

[17]:
```
log_reg = LogisticRegression(random_state = 42)
```

[18]:
```
log_reg.fit(X_train_preprocessed,y_train)
```

[18]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=42, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

[19]:
```
score = log_reg.score(X_val_preprocessed,y_val)
print("The validation score for Logistic Regression is: " + str(score))
```

```
The validation score for Logistic Regression is: 0.7375
```

[20]:
```
model_names = ["LogisticRegression", "LinearSVC", "KNeighboursClassifier"]
model_list = [ LogisticRegression(), LinearSVC(), KNeighborsClassifier()]
```

[21]:
```
preprocess = make_column_transformer((OneHotEncoder(handle_unknown="ignore"),␣
 ↪categorical_columns_index))

scores = []
for i in range(len(model_list)):
    score = cross_val_score(make_pipeline(preprocess, model_list[i]),␣
 ↪X_train_val, y_train_val, cv=5)
    scores.append(np.mean(score))

for i in range(len(model_names)):
    print("The validation score for " + model_names[i] +" is " + str(scores[i]))
```

```
The validation score for LogisticRegression is 0.7374332981757099
The validation score for LinearSVC is 0.7336755342005546
The validation score for KNeighboursClassifier is 0.6974330051955155
```

[22]:
```
preprocess = make_column_transformer((OneHotEncoder(handle_unknown="ignore"),␣
 ↪categorical_columns_index),(StandardScaler(), continuous_columns_index))

scores = []
for i in range(len(model_list)):
    score = cross_val_score(make_pipeline(preprocess, model_list[i]),␣
 ↪X_train_val, y_train_val, cv=5)
    scores.append(np.mean(score))

for i in range(len(model_names)):
    print("The validation score for " + model_names[i] +" is " + str(scores[i]))
```

```
The validation score for LogisticRegression is 0.7436911598109301
The validation score for LinearSVC is 0.7411990214461502
The validation score for KNeighboursClassifier is 0.7361833958357749
```

We can see that the scaling helps a lot in the case of KNearestNeighbours classifier which is quite intutive from the fact that it relies upon the distances and use them intensively for the prediction of the class.

The results for Linear SVM and Logistic Regression also are improved but not as much as that of KNearestNeighbours Classifier.

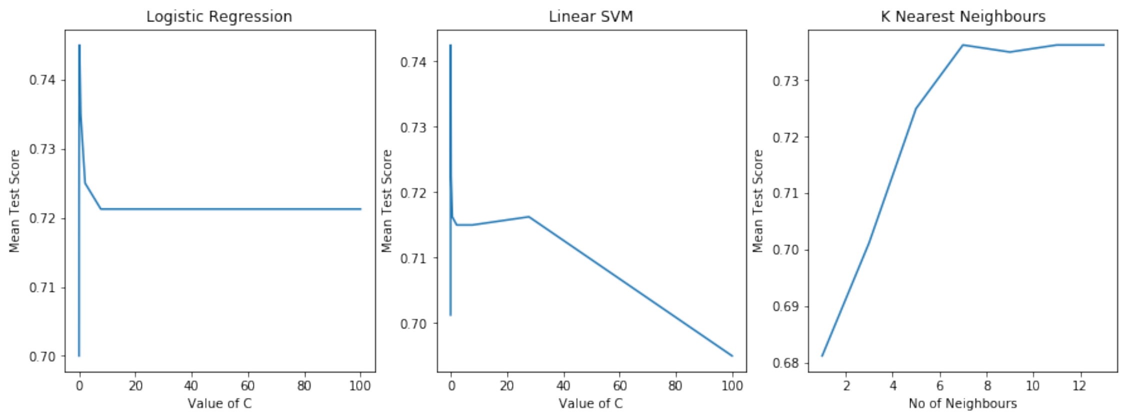
[23]:
```
preprocess = make_column_transformer((OneHotEncoder(handle_unknown="ignore"),␣
↪categorical_columns_index),(StandardScaler(), continuous_columns_index))

pipe = Pipeline([('preprocess', preprocess),
                 ('classfier', LogisticRegression())])

param_grid = [{'classfier': [LogisticRegression(random_state=42)],
               'classfier__C': np.logspace(-3, 2, 10)},
              {'classfier': [LinearSVC(random_state=42)],
               'classfier__C': np.logspace(-3, 2, 10)},
              {'classfier': [KNeighborsClassifier()],
               'classfier__n_neighbors': range(1,15,2)}
             ]
grid = GridSearchCV(pipe, param_grid)
grid.fit(X_train_val, y_train_val)
best_score = grid.score(X_test, y_test)

print("The best score:" + str(best_score))
print("The best method along with its parameter values is: " + str(grid.
↪best_params_))
```

```
The best score:0.79
The best method along with its parameter values is: {'classfier':
LogisticRegression(C=0.1668100537200059, class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=42, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False), 'classfier__C': 0.1668100537200059}
```

[24]:
```
plots = pd.DataFrame(grid.cv_results_)
plots["LR"]= [isinstance(plots["param_classfier"][i],LogisticRegression) for i␣
↪in range(plots.shape[0])]
plots["SVM"] = [isinstance(plots["param_classfier"][i],LinearSVC) for i in␣
↪range(plots.shape[0])]
plots["KNN"] = [isinstance(plots["param_classfier"][i],KNeighborsClassifier)␣
↪for i in range(plots.shape[0])]
```

[25]:
```
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

axes[0].
↪plot(plots[plots["LR"]]["param_classfier__C"],plots[plots["LR"]]["mean_test_score"])
axes[1].
↪plot(plots[plots["SVM"]]["param_classfier__C"],plots[plots["SVM"]]["mean_test_score"])
```

```
axes[2].
↪plot(plots[plots["KNN"]]["param_classfier__n_neighbors"],plots[plots["KNN"]]["mean_test_sco
axes[0].set_xlabel("Value of C")
axes[0].set_ylabel("Mean Test Score")
axes[0].set_title("Logistic Regression")

axes[1].set_xlabel("Value of C")
axes[1].set_ylabel("Mean Test Score")
axes[1].set_title("Linear SVM")

axes[2].set_xlabel("No of Neighbours")
axes[2].set_ylabel("Mean Test Score")
axes[2].set_title("K Nearest Neighbours")

plt.show()
```



The results as can be seen has improved from all of the three case having scores (0.74, 0.74, 0.73) respectively. While after the gridsearch it can be seen that the best model is found to have a score of 0.79.

[26]:
```
preprocess = make_column_transformer((OneHotEncoder(handle_unknown="ignore"),␣
↪categorical_columns_index),(StandardScaler(), continuous_columns_index))

pipe = Pipeline([('preprocess', preprocess),
                 ('classfier', LogisticRegression())])

param_grid = [{'classfier': [LogisticRegression(random_state=42)],
               'classfier__C': np.logspace(-3, 2, 10)},
              {'classfier': [LinearSVC(random_state=42)],
               'classfier__C': np.logspace(-3, 2, 10)},
              {'classfier': [KNeighborsClassifier()],
               'classfier__n_neighbors': range(1,15,2)}
             ]
```

```
grid = GridSearchCV(pipe, param_grid,cv = KFold(shuffle=True ,␣
↪random_state=100))
grid.fit(X_train_val, y_train_val)
best_score = grid.score(X_test, y_test)

print("The best score (on the test):" + str(best_score))
print("The best method along with its parameter values is: " + str(grid.
↪best_params_))
```

```
The best score (on the test):0.77
The best method along with its parameter values is: {'classfier':
LogisticRegression(C=0.046415888336127795, class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=42, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False), 'classfier__C': 0.046415888336127795}
```

[27]:
```
preprocess = make_column_transformer((OneHotEncoder(handle_unknown="ignore"),␣
↪categorical_columns_index),(StandardScaler(), continuous_columns_index))

pipe = Pipeline([('preprocess', preprocess),
                 ('classfier', LogisticRegression())])

param_grid = [{'classfier': [LogisticRegression(random_state=42)],
               'classfier__C': np.logspace(-3, 2, 10)},
              {'classfier': [LinearSVC(random_state=42)],
               'classfier__C': np.logspace(-3, 2, 10)},
              {'classfier': [KNeighborsClassifier()],
               'classfier__n_neighbors': range(1,15,2)}
             ]
grid = GridSearchCV(pipe, param_grid,cv = KFold(shuffle=True , random_state=42))
grid.fit(X_train_val, y_train_val)
best_score = grid.score(X_test, y_test)

print("The best score (on the test):" + str(best_score))
print("The best method along with its parameter values is: " + str(grid.
↪best_params_))
```

```
The best score (on the test):0.79
The best method along with its parameter values is: {'classfier':
LogisticRegression(C=0.1668100537200059, class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=42, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False), 'classfier__C': 0.1668100537200059}
```

```
[28]: X_train_val, X_test, y_train_val, y_test = train_test_split(X_df, y_df,␣
       ↪test_size=0.20, random_state=83)

      preprocess = make_column_transformer((OneHotEncoder(handle_unknown="ignore"),␣
       ↪categorical_columns_index),(StandardScaler(), continuous_columns_index))

      pipe = Pipeline([('preprocess', preprocess),
                       ('classfier', LogisticRegression())])

      param_grid = [{'classfier': [LogisticRegression(random_state=42)],
                     'classfier__C': np.logspace(-3, 2, 10)},
                    {'classfier': [LinearSVC(random_state=42)],
                     'classfier__C': np.logspace(-3, 2, 10)},
                    {'classfier': [KNeighborsClassifier()],
                     'classfier__n_neighbors': range(1,15,2)}
                   ]
      grid = GridSearchCV(pipe, param_grid,cv = KFold(shuffle=True , random_state=42))
      grid.fit(X_train_val, y_train_val)
      best_score = grid.score(X_test, y_test)

      print("The best score (on the test):" + str(best_score))
      print("The best method along with its parameter values is: " + str(grid.
       ↪best_params_))
```

```
The best score (on the test):0.72
The best method along with its parameter values is: {'classfier':
LogisticRegression(C=2.1544346900318843, class_weight=None, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=42, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False), 'classfier__C': 2.1544346900318843}
```

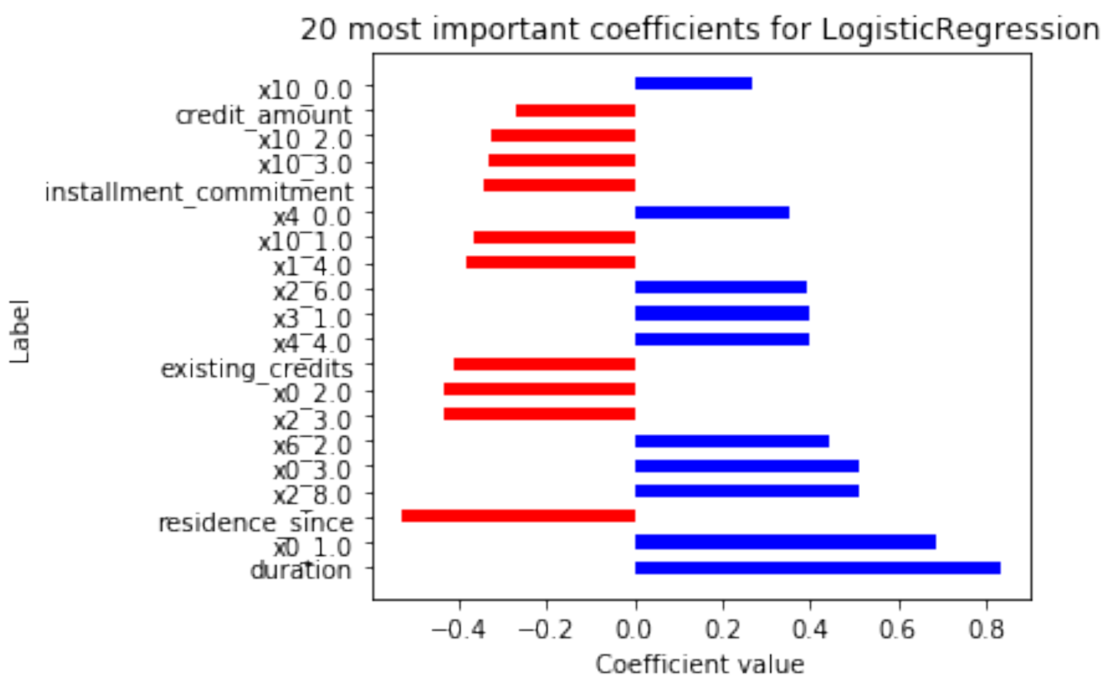
- The above 3 blocks of code show that the results change with change in value of random seed. These changes are evident from change in random seed's value at both the positions (i.e.) while performing the train_test split as well as one used in the KFold method to indicate random shuffling.

```
[29]: preprocess = make_column_transformer((OneHotEncoder(handle_unknown="ignore"),␣
       ↪categorical_columns_index),(StandardScaler(), continuous_columns_index))

      pipe1 = Pipeline([('preprocess', preprocess),
                        ('classifier', LogisticRegression(random_state=42))])

      param_grid = [{'classifier__C': np.logspace(-3, 2, 10)}]
      grid = GridSearchCV(pipe1, param_grid)
      grid.fit(X_train_val, y_train_val)
      best_score = grid.score(X_test, y_test)
```

```
print("The best parameter values for Logistic Regression: " + str(grid.
↪best_params_))
```

```
The best parameter values for Logistic Regression: {'classifier__C':
0.1668100537200059}
```

[30]:
```
pipe = Pipeline([('scaler', preprocess),('classifier',␣
↪LogisticRegression(C=grid.best_params_["classifier__C"]))])
pipe.fit(X_train_val,y_train_val)

coefficients=pipe.get_params()['classifier'].coef_
cat_names= list(preprocess.named_transformers_['onehotencoder'].
↪get_feature_names())
features = continuous_columns.copy()
features.extend(cat_names)


coefficient_list = list(coefficients[0])

# combined_list = [(i,j) for i,j in zip(coefficient_list,features)]
# sorted_combination =sorted(combined_list, key = lambda x:␣
↪abs(x[0]),reverse=True)
# top_20 = sorted_combination[:20]

fig, axes = plt.subplots()
for coef in sorted(coefficient_list,key = lambda x: abs(x),reverse=True)[:20]:
    plt.barh(features[coefficient_list.index(coef)], coef, height=.5, color=plt.
↪cm.bwr_r(np.sign(coef)))

plt.xlabel("Coefficient value")
plt.ylabel("Label")
plt.title("20 most important coefficients for LogisticRegression")
plt.tight_layout()
```

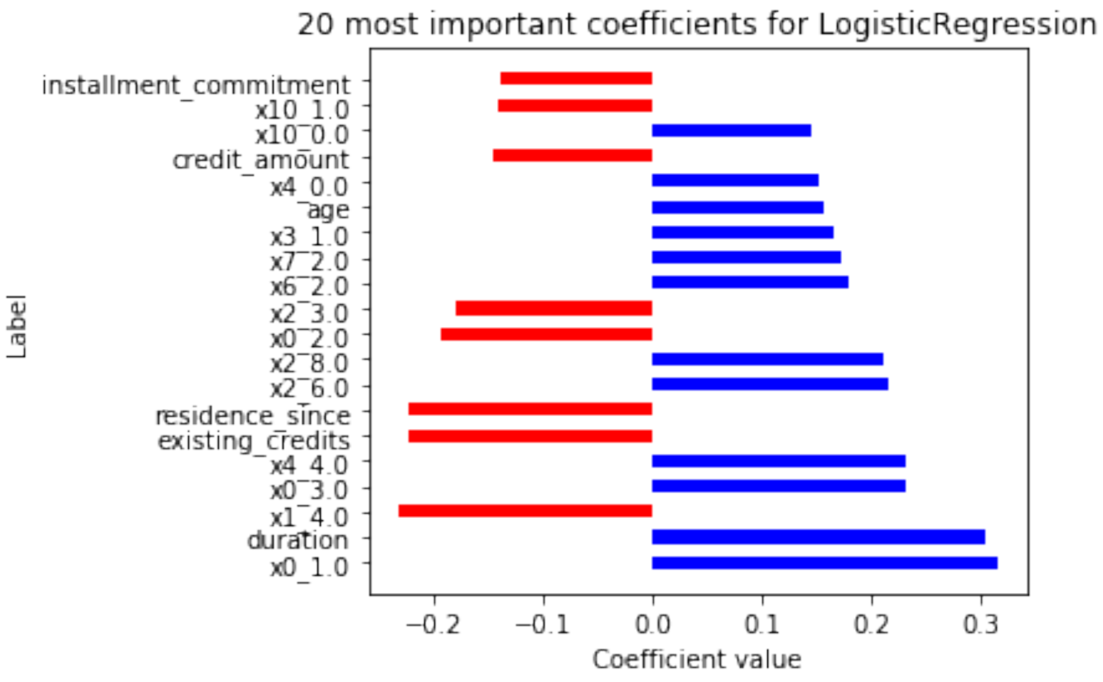


[31]:
```
preprocess = make_column_transformer((OneHotEncoder(handle_unknown="ignore"),␣
 ↪categorical_columns_index),(StandardScaler(), continuous_columns_index))

pipe1 = Pipeline([('preprocess', preprocess),
                  ('classifier', LinearSVC(random_state=42))])

param_grid = [{'classifier__C': np.logspace(-3, 2, 10)}]
grid = GridSearchCV(pipe1, param_grid)

grid.fit(X_train_val, y_train_val)
best_score = grid.score(X_test, y_test)

print("The best parameter values for Linear SVM: " + str(grid.best_params_))
```

```
The best parameter values for Linear SVM: {'classifier__C':
0.046415888336127795}
```

[32]:
```
pipe = Pipeline([('scaler', preprocess),('classifier', LinearSVC(C=grid.
 ↪best_params_["classifier__C"]))])
pipe.fit(X_train_val,y_train_val)

coefficients=pipe.get_params()['classifier'].coef_
cat_names= list(preprocess.named_transformers_['onehotencoder'].
 ↪get_feature_names())
features = continuous_columns.copy()
```

```
features.extend(cat_names)

coefficient_list = list(coefficients[0])

# combined_list = [(i,j) for i,j in zip(coefficient_list,features)]
# sorted_combination =sorted(combined_list, key = lambda x:␣
 ↪abs(x[0]),reverse=True)
# top_20 = sorted_combination[:20]

fig, axes = plt.subplots()
for coef in sorted(coefficient_list,key = lambda x: abs(x),reverse=True)[:20]:
    plt.barh(features[coefficient_list.index(coef)], coef, height=.5, color=plt.
 ↪cm.bwr_r(np.sign(coef)))

plt.xlabel("Coefficient value")
plt.ylabel("Label")
plt.title("20 most important coefficients for LogisticRegression")
plt.tight_layout()
```



pds2136_Task2

February 19, 2020

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import␣
 ↪train_test_split,cross_val_score,GridSearchCV,KFold
from sklearn.preprocessing import OneHotEncoder,StandardScaler
from sklearn.compose import make_column_transformer
from sklearn.pipeline import make_pipeline, Pipeline
from category_encoders import TargetEncoder
from sklearn.impute import SimpleImputer

from sklearn.linear_model import LinearRegression,Ridge,Lasso,ElasticNet
```

[3]:
```
data = pd.read_csv("data.csv")
data = data.drop(["date","country","street"],axis=1)
```

- Dropping the “Date”, “Country” and “Street” column as they don’t add any important useful information to the model building.

[4]:
```
X = data.drop("price", axis=1)
y = pd.DataFrame(data["price"])
```

[5]:
```
categorical = X.dtypes == object
X = X.drop(y[y.price <= 0].index)
y = y.drop(y[y.price <= 0].index)
```

- Waterfront: A dummy variable for whether the apartment was overlooking the waterfront or not
- View: An index from 0 to 4 of how good the view of the property was
- Condition: An index from 1 to 5 on the condition of the apartment

The above variables appear to be be nominal. Here, we consider those as continuous.

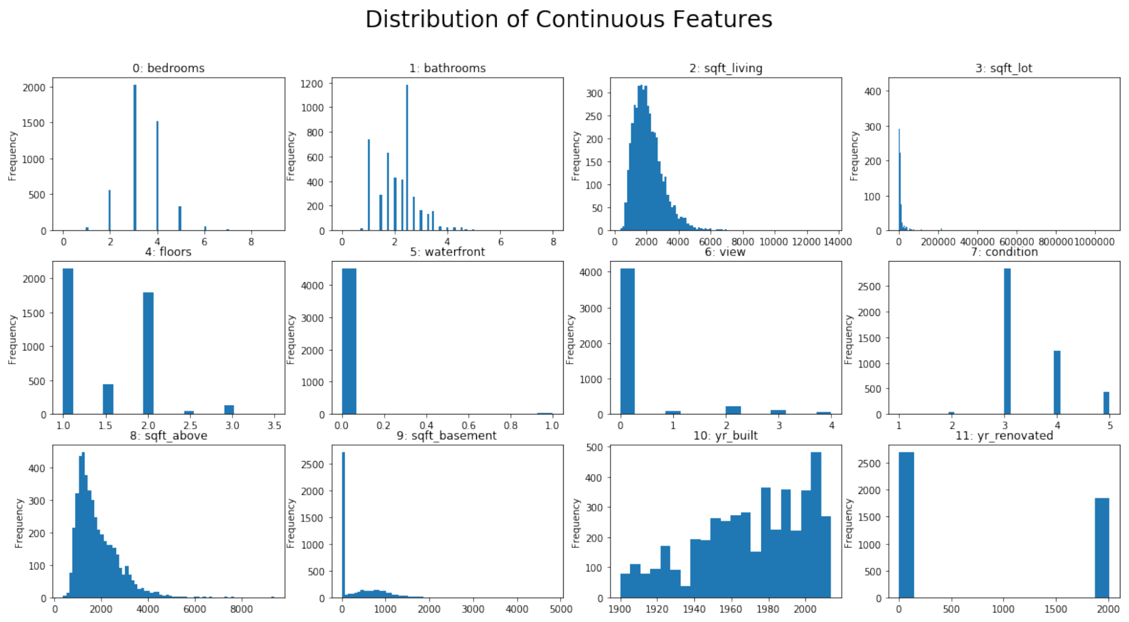
[6]:
```
features = list(X.columns)
```

[7]:
```
categorical_columns = [i for i,v in categorical.items() if v]
continuous_columns = [i for i,v in categorical.items() if not v]

categorical_columns_index = [i for i,v in enumerate(features) if(v in␣
↪categorical_columns)]
continuous_columns_index = [i for i,v in enumerate(features) if(v in␣
↪continuous_columns)]
```

[8]:
```
print("- Following is the list of continuous features:")
print(continuous_columns)
print()
print("- Following is the list of categorical features:")
print(categorical_columns)
```

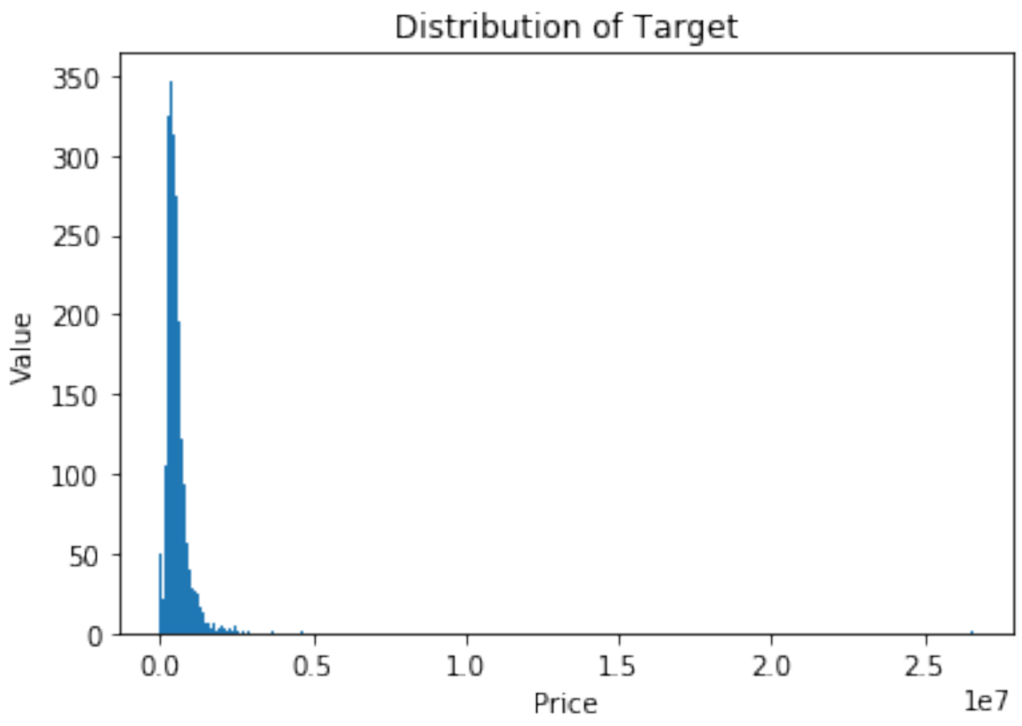
```
- Following is the list of continuous features:
['bedrooms', 'bathrooms', 'sqft_living', 'sqft_lot', 'floors', 'waterfront',
'view', 'condition', 'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated']

- Following is the list of categorical features:
['city', 'statezip']
```

[9]:
```
fig, axes = plt.subplots(3, 4, figsize=(20, 10))
for i, ax in enumerate(axes.ravel()):
    ax.hist(X.iloc[:,i],bins="auto")
    ax.set_title("{}: {}".format(i, continuous_columns[i]))
    ax.set_ylabel("Frequency")
fig.suptitle("Distribution of Continuous Features", fontsize=24)
plt.show()
```



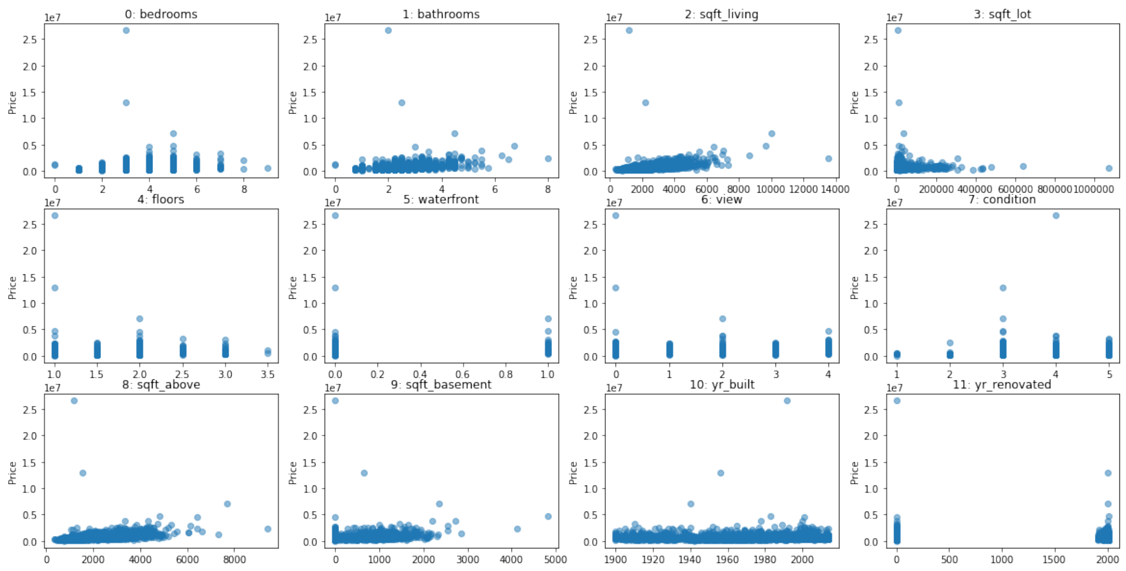
- From the above plots it can be seen that the year renovated is having a lot of values at 0. Similar is the case for Squarefoot_basement and squarefoot lot. Thus, the 0 value seems to represent the missing values in the Dataframe and they need to be handelled.
- Also, the above plots show the nominal distributions in the case of waterdront, view, condition.

[10]:
```
plt.hist(data["price"], bins="auto")
plt.xlabel("Price")
plt.ylabel("Value")
plt.title("Distribution of Target")
plt.show()
```



- The above plot shows that the dependent variable is sked a lot towards right side. The log transformation can be used to make it centered and have a uniformly distributed space.

[11]:
```
fig, axes = plt.subplots(3, 4, figsize=(20, 10))
for i, ax in enumerate(axes.ravel()):
    ax.plot(X.iloc[:,i], y, 'o', alpha=.5)
    ax.set_title("{}: {}".format(i, continuous_columns[i]))
    ax.set_ylabel("Price")
plt.show()
```



[12]:
```
X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.
↪20, random_state=42)
```

[13]:
```
preprocess = make_column_transformer((TargetEncoder(),␣
↪categorical_columns_index),(SimpleImputer(missing_values=0,␣
↪strategy="median"), continuous_columns_index))
```

[14]:
```
pipe1 = Pipeline([('preprocess', preprocess),
                 ('regressor', LinearRegression())])
pipe2 = Pipeline([('preprocess', preprocess),
                 ('regressor', Ridge())])
pipe3 = Pipeline([('preprocess', preprocess),
                 ('regressor', Lasso())])
pipe4 = Pipeline([('preprocess', preprocess),
                 ('regressor', ElasticNet())])

pipes=[pipe1,pipe2,pipe3,pipe4]
model_names=["Linear Regression","Ridge Regression","Lasso Regression","Elastic␣
↪Net"]
scores=[]

for i in pipes:
    score = cross_val_score(i, X_train_val, y_train_val, cv=5)
    scores.append(np.mean(score))

for i in range(len(model_names)):
    print("The validation score for " + model_names[i] +" is " + str(scores[i]))
```

```
The validation score for Linear Regression is 0.5556733214589125
The validation score for Ridge Regression is 0.5556827318391002
The validation score for Lasso Regression is 0.5556743727451792
The validation score for Elastic Net is 0.5480590225577192
```

[15]:
```
scaling = make_column_transformer((StandardScaler(),␣
↪continuous_columns_index),remainder = "passthrough")
pipe1 = Pipeline([('preprocess', preprocess),
                  ('Scaler',scaling),
                 ('regressor', LinearRegression())])
pipe2 = Pipeline([('preprocess', preprocess),
                  ('Scaler',scaling),
                 ('regressor', Ridge())])
pipe3 = Pipeline([('preprocess', preprocess),
                  ('Scaler',scaling),
                 ('regressor', Lasso())])
pipe4 = Pipeline([('preprocess', preprocess),
                  ('Scaler',scaling),
                 ('regressor', ElasticNet())])

pipes=[pipe1,pipe2,pipe3,pipe4]
model_names=["Linear Regression","Ridge Regression","Lasso Regression","Elastic␣
↪Net"]
scores=[]

for i in pipes:
    score = cross_val_score(i, X_train_val, y_train_val, cv=5)
    scores.append(np.mean(score))

for i in range(len(model_names)):
    print("The validation score for " + model_names[i] +" is " + str(scores[i]))
```

```
The validation score for Linear Regression is 0.5556733214588785
The validation score for Ridge Regression is 0.5557223941572328
The validation score for Lasso Regression is 0.5556751755862074
The validation score for Elastic Net is 0.550711374715979
```

- Scaling the data with the help of pipeline as shown above does help to increase the score and imporve the performance of models(in all cases apart from the normal linear regression(OLS). The imporvement in the particualr case is not that signoificant in the case of Ridge and Lasso regression but as can be seen that in the case of Elastic Net is quite significant.

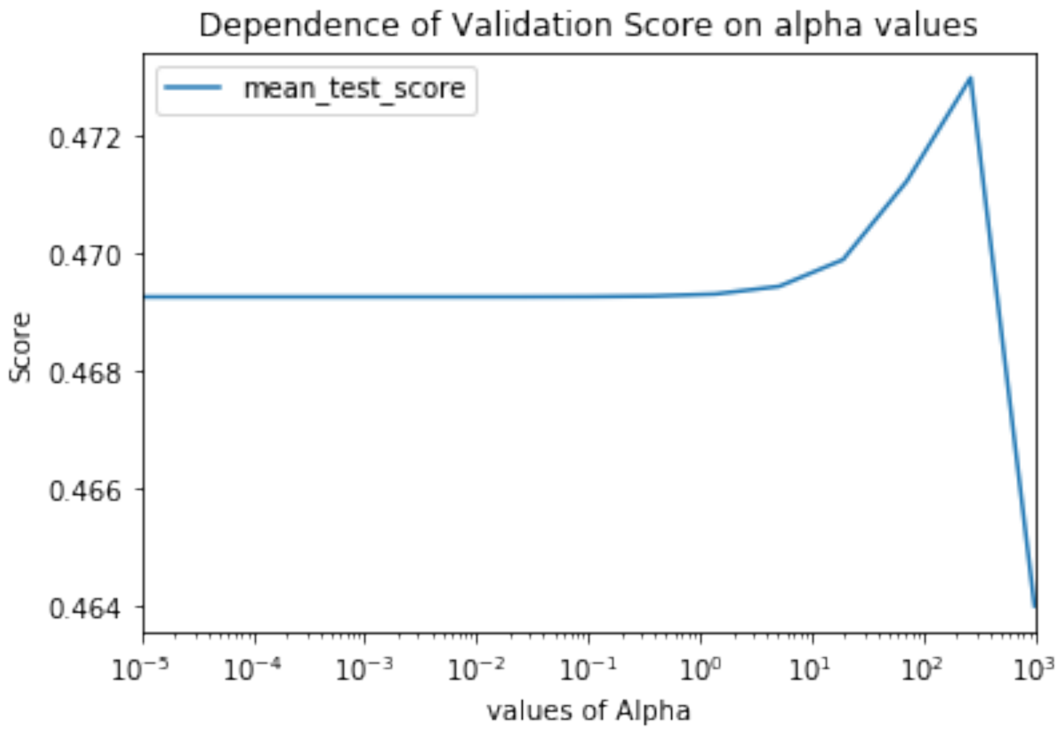
[16]:
```
param_grid_ridge = [{'regressor__alpha': np.logspace(-5, 3, 15)}]
param_grid_lasso = [{'regressor__alpha': np.logspace(-5, 3, 15)}]
param_grid_elastic = [{'regressor__l1_ratio':np.logspace(-5,1,15) }]
```

[17]:
```
grid_ridge = GridSearchCV(pipe2, param_grid_ridge)
grid_ridge.fit(X_train_val, y_train_val)
```

```
best_score = grid_ridge.score(X_test, y_test)
print("The best score for Ridge Regression is: " + str(best_score) + " with the␣
 ↪parameter value: " + str(grid_ridge.best_params_))
```

```
The best score for Ridge Regression is: 0.6736647078680223 with the parameter
value: {'regressor__alpha': 268.26957952797216}
```

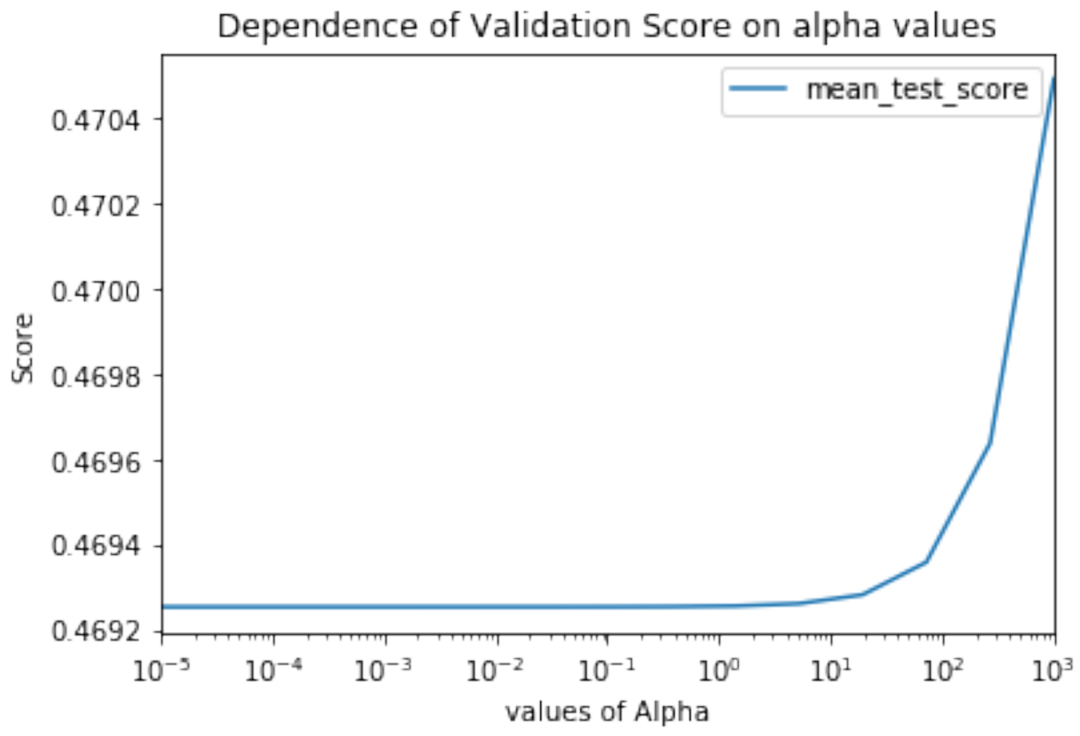
[18]:
```
results = pd.DataFrame(grid_ridge.cv_results_)
results.plot('param_regressor__alpha', 'mean_test_score', ax=plt.gca())
plt.xscale("log")
plt.title("Dependence of Validation Score on alpha values")
plt.xlabel("values of Alpha")
plt.ylabel("Score")
plt.show()
```



[19]:
```
grid_lasso = GridSearchCV(pipe3, param_grid_lasso)
grid_lasso.fit(X_train_val, y_train_val)
best_score = grid_lasso.score(X_test, y_test)
print("The best score for Lasso Regression is: " + str(best_score) + " with the␣
 ↪parameter value: " + str(grid_lasso.best_params_))
```

```
The best score for Lasso Regression is: 0.6683617421816039 with the parameter
value: {'regressor__alpha': 1000.0}
```

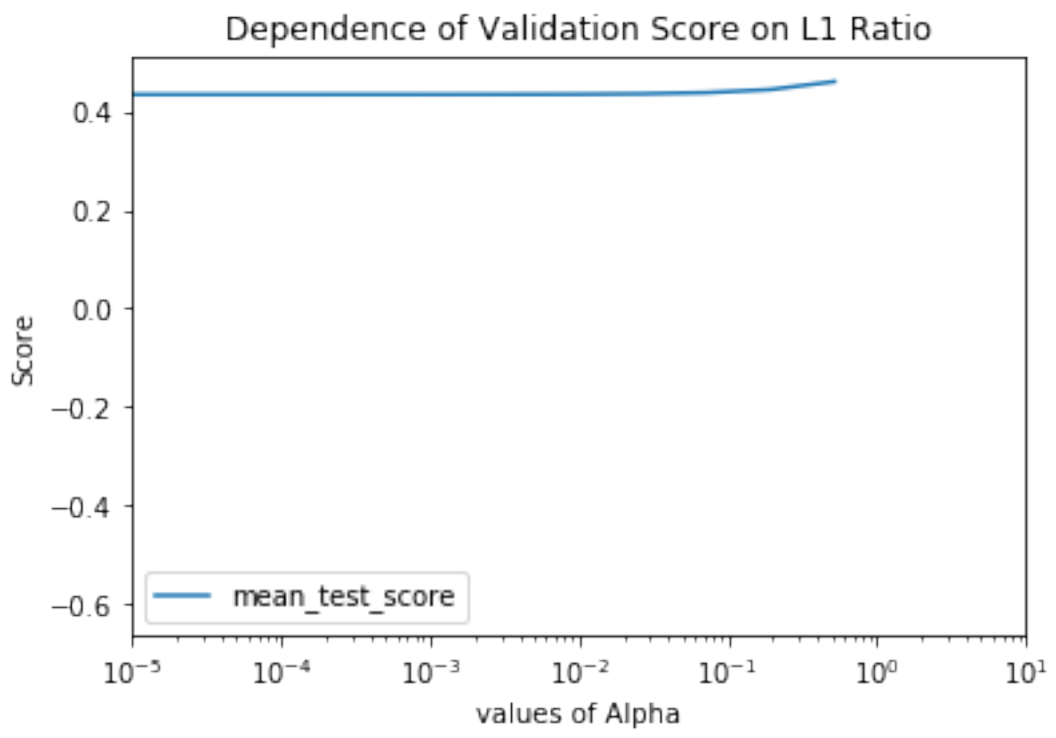
[20]:
```
results = pd.DataFrame(grid_lasso.cv_results_)
results.plot('param_regressor__alpha', 'mean_test_score', ax=plt.gca())
plt.xscale("log")
plt.title("Dependence of Validation Score on alpha values")
plt.xlabel("values of Alpha")
plt.ylabel("Score")
plt.show()
```



[21]:
```
grid_elastic = GridSearchCV(pipe4, param_grid_elastic)
grid_elastic.fit(X_train_val, y_train_val)
best_score = grid_elastic.score(X_test, y_test)
print("The best score for Elastic Net is: " + str(best_score) + " with the␣
 ↪parameter value: " + str(grid_elastic.best_params_))
```

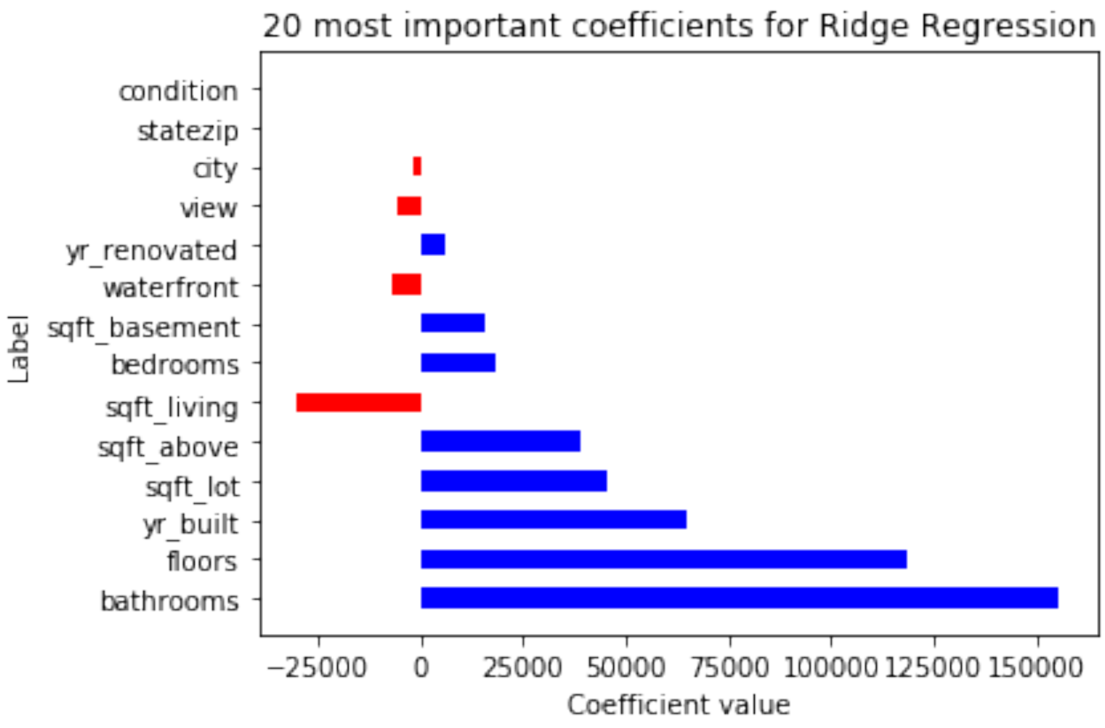
```
The best score for Elastic Net is: 0.6617189908401967 with the parameter value:
{'regressor__l1_ratio': 0.5179474679231213}
```

[22]:
```
results = pd.DataFrame(grid_elastic.cv_results_)
results.plot('param_regressor__l1_ratio', 'mean_test_score', ax=plt.gca())
plt.xscale("log")
plt.title("Dependence of Validation Score on L1 Ratio")
plt.xlabel("values of Alpha")
plt.ylabel("Score")
plt.show()
```

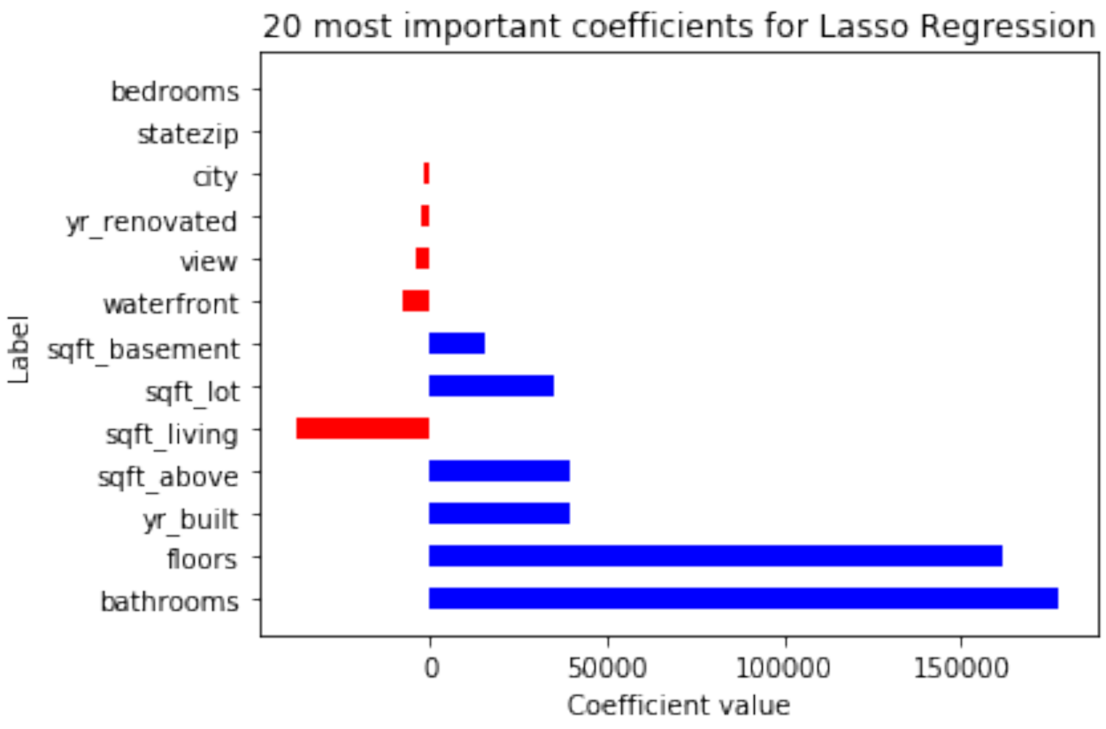


The results in all 3 cases improve with the help of GRID Search as it helps us tune the hyperparameter (Alpha in the case of Ridge and Lasso whereas L1-Ratio in the case Elastic Net). Particularly, - In ridge regression score improves from 0.5557223941572328 to 0.6736647078680223 - In Lasso regression score improves from 0.5556751755862074 to 0.6683617421816039 - In Elastic Net score improves from 0.550711374715979 to 0.6617189908401967

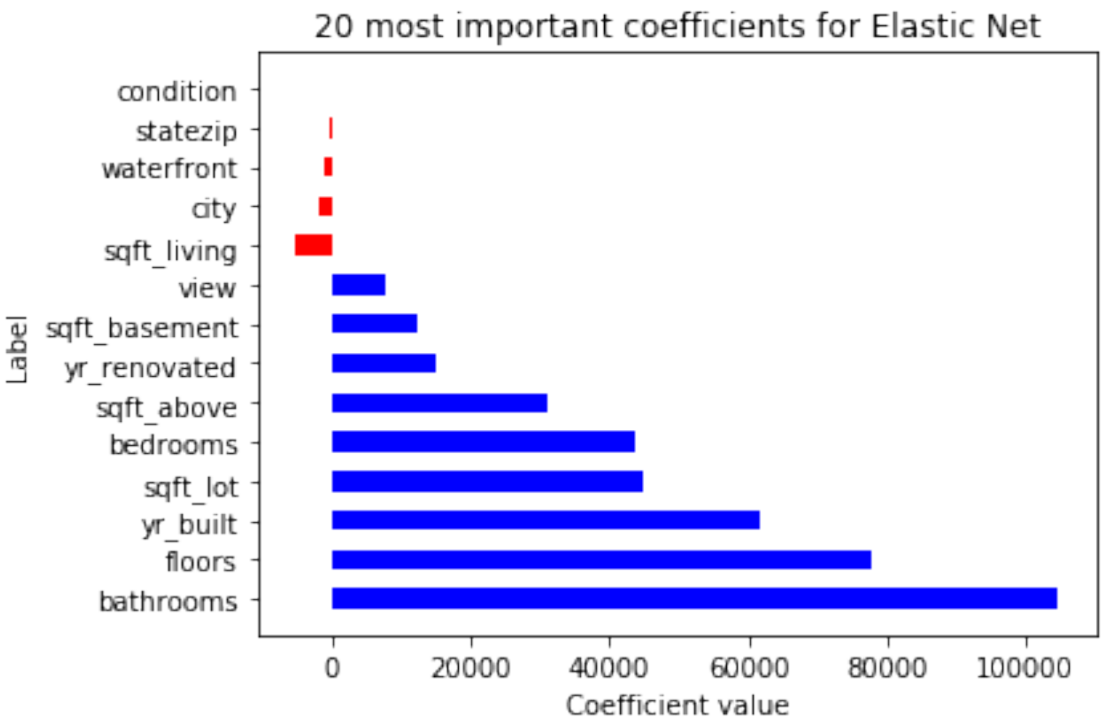
[23]:
```
coefficients=grid_ridge.best_estimator_.get_params()['regressor'].coef_
features = continuous_columns.copy()
features.extend(categorical_columns)

coefficient_list = list(coefficients[0])

fig, axes = plt.subplots()
for coef in sorted(coefficient_list,key = lambda x: abs(x),reverse=True)[:20]:
    plt.barh(features[coefficient_list.index(coef)], coef, height=.5, color=plt.
 ↪cm.bwr_r(np.sign(coef)))

plt.xlabel("Coefficient value")
plt.ylabel("Label")
plt.title("20 most important coefficients for Ridge Regression")
plt.tight_layout()
plt.show()
```



[24]:
```
coefficients=grid_lasso.best_estimator_.get_params()['regressor'].coef_
features = continuous_columns.copy()
features.extend(categorical_columns)

coefficient_list = list(coefficients)

fig, axes = plt.subplots()
for coef in sorted(coefficient_list,key = lambda x: abs(x),reverse=True)[:20]:
    plt.barh(features[coefficient_list.index(coef)], coef, height=.5, color=plt.
 ↪cm.bwr_r(np.sign(coef)))

plt.xlabel("Coefficient value")
plt.ylabel("Label")
plt.title("20 most important coefficients for Lasso Regression")
plt.tight_layout()
plt.show()
```



[25]:
```
coefficients=grid_elastic.best_estimator_.get_params()['regressor'].coef_
features = continuous_columns.copy()
features.extend(categorical_columns)

coefficient_list = list(coefficients)

fig, axes = plt.subplots()
for coef in sorted(coefficient_list,key = lambda x: abs(x),reverse=True)[:20]:
    plt.barh(features[coefficient_list.index(coef)], coef, height=.5, color=plt.
 ↪cm.bwr_r(np.sign(coef)))

plt.xlabel("Coefficient value")
plt.ylabel("Label")
plt.title("20 most important coefficients for Elastic Net")
plt.tight_layout()
plt.show()
```



From the above figures it can be seen that all of the 3 models agree on the fact that “Yr_built”, “bathrooms” and “floor” are the top-3 features.

They slighltly deviate from each other in the sense that Elastic rate “View” to have positive coefficient while other two claim it to be “negative”. Apart from that the coefficients for other features differ in magnitude but having similar sign showing the simialr impact over the dependent variable in all the cases.