Data Science Project 2- Alaukik

Performing model evaluation, model selection and feature selection in both a regression and classification setting.

The data is a small set of home sales data from as we might see on a real-estate website.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pylab as plt

%matplotlib
np.random.seed(123)
```

```
Using matplotlib backend: MacOSX
```

Regression

Here, we will try to predict a real value home sale price using several models.

In [2]:
```
df = pd.read_csv('../data/house_sales_subset.csv')

# Extracting the dataframe X which should contains the first 5 columns
:
# SqFtTotLiving_x1000', 'SqFtLot_x1000', 'SqFtDriveway_x1000', 'Bathro
oms', 'Bedrooms'
X = df.iloc[:, 0:5]
# Extract the series y_r which should contain only the last column Adj
SalePrice_x100000
y_r = df.iloc[:, -1]
```

In [3]:
```
# Creating a held-aside set.


from sklearn.model_selection import train_test_split

X_train_r, X_test_r, y_train_r, y_test_r = train_test_split(X, y_r, te
st_size=0.2, random_state=42)

print(f'{len(y_test_r)/ len(y_r)}')
```

```
0.2
```

In [4]:
```
# Creating a Dummy Regressior for baseline comparison


# Importing the DummyRegressor model from sklearn
from sklearn.dummy import DummyRegressor

# Instantiating a dummy model using strategy="median"
dummy_r = DummyRegressor(strategy="median")

# Training the dummy model on the training set created above
linr = dummy_r.fit(X_train_r,y_train_r)

# Calculating and print the training set R2 score of the trained model
.
dummy_r_training_r2= dummy_r.score(X_train_r,y_train_r)

print('dummy training set R2: {:.2f}'.format(dummy_r_training_r2))
```

```
dummy training set R2: -0.06
```

In [5]:
```
# Using 5-fold Cross Validation to get a set of negative-MSE scores

from sklearn.model_selection import cross_val_score

# Generating 5-fold cross valication neg_mean_squared_error scores
#    for the Dummy model on the training set.
dummy_r_negmse_cvscores = cross_val_score(linr, X_train_r, y_train_r,
cv=5,
                         scoring='neg_mean_squared_error')

print(dummy_r_negmse_cvscores)
```

```
[-5.05363975 -4.28957165 -6.09214843 -6.16181789 -4.46183041]
```

In [6]:
```
# Since we'll need to convert from negative-MSE to RMSE several times, we will
# write a function that takes in a list of negative-MSE scores and returns positive mean RMSE and 2 times the standard deviation


def negmse_to_rmse(negmse_cvscores):

    mse_cvscores = abs(negmse_cvscores)

    rmse_cvscores = np.sqrt(mse_cvscores)

    rmse_mean = np.mean(rmse_cvscores)

    rmse_2std = 2 * np.std(rmse_cvscores)

    return(rmse_mean,rmse_2std)
```

In [7]:
```
# Using our negmse_to_rmse function to calculate mean-RMSE
#     and standard deviations for the dummy model.

# Passing dummy_r_negmse_cvscores to our function and capture the output
dummy_r_rmse,dummy_r_rmse_2std = negmse_to_rmse(dummy_r_negmse_cvscores)

# Printing out the mean RMSE and 2 standard variations for the dummy model
print('dummy mean cv RMSE: {:.2f} +- {:.2f}'.format(dummy_r_rmse,dummy_r_rmse_2std))
```

```
dummy mean cv RMSE: 2.28 +- 0.35
```

In [8]:
```
# Importing the Linear Regression model and calculating mean RMSE usin
g 5-fold Cross Validation

# Importing the LinearRegression model from sklearn
from sklearn.linear_model import LinearRegression

# Generating 5-fold cv neg_mean_squared_error scores
#   for the LinearRegression model with default settings
#   on the training set.
linre = LinearRegression().fit(X_train_r,y_train_r)
lr_negmse_cvscores = cross_val_score(linre, X_train_r, y_train_r, cv=5
, scoring='neg_mean_squared_error')
# Using the function we wrote above to get mean RMSE and 2 standard de
viations for LinearRegression.
lr_rmse,lr_rmse_2std = negmse_to_rmse(lr_negmse_cvscores)
# Printing out the mean RMSE and 2 standard variations for LinearRegre
ssion
print('lr mean cv RMSE: {:.2f} +- {:.2f}'.format(lr_rmse,lr_rmse_2std)
)
```

```
lr mean cv RMSE: 1.54 +- 0.20
```

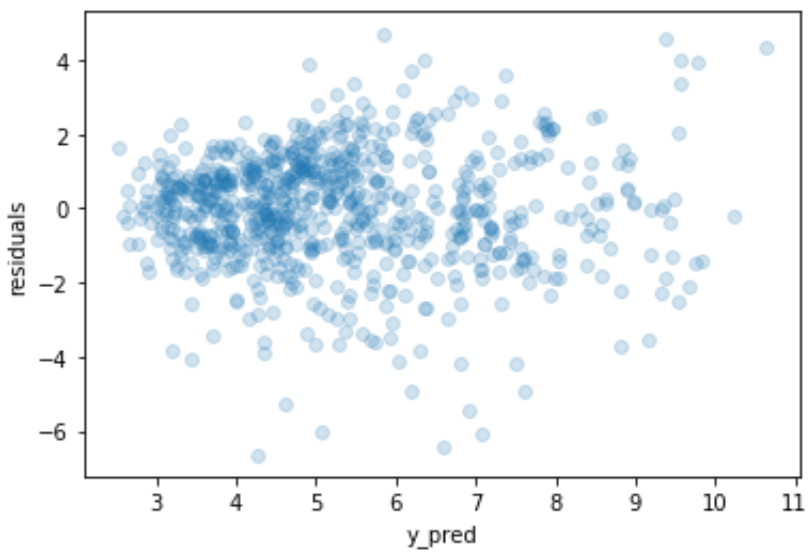
In [9]:
```
# Ploting the residuals of a Linear Regression model

# Instantiating and retraining a linear regression model on the entire
training set.
lr = LinearRegression().fit(X_train_r,y_train_r)

# Generating predictions y_pred, again using the training set.
y_pred = lr.predict(X_train_r)

# Calculating residuals
residuals = y_pred - y_train_r

# Ploting predictions (x-axis) vs residuals (y-axis) using plt.scatter
()
#    In scatter set alpha=0.2 to make the markers somewhat transparent
.
#    Setting axis/label names appropriately ('y_pred' and 'residuals')
plt.scatter(y_pred, residuals, alpha=0.2)
plt.xlabel("y_pred")
plt.ylabel("residuals")
plt.show()
```



The residuals appear fairly normal around 0 across the range of y_pred

ElasticNet HyperParameter Tuning

In [10]:
```
# 10. Using GridSearch to choose an optimal hyperparamter setting for
ElasticNet

# Importing ElasticNet and GridSearchCV from sklearn
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import ElasticNet

# Performing GridSearch over potential settings of the l1_ratio = [.1,
.5,.9,1]

params = {'l1_ratio': [.1,.5,.9,1] }
gscv = GridSearchCV(ElasticNet(),params,cv=5)
gscv.fit(X_train_r,y_train_r)

# Printing out the best parameter setting found using grid search and
the best parameter setting found
print('gscv best params: {}'.format(gscv.best_params_))
```

```
gscv best params: {'l1_ratio': 0.1}
```

In [11]:
```
# Calculating the average RMSE for the ElasticNet model using 5-fold C
ross Validation

# Instantiating a new ElasticNet model with the optimal l1_ratio found
above.
en = ElasticNet(l1_ratio=0.1)

# Generating 5-fold cv neg_mean_squared_error scores
#   for the instantiated ElasticNet model on the training set.
en_negmse_cvscores = cross_val_score(en, X_train_r, y_train_r, cv=5, s
coring='neg_mean_squared_error')

# Using the function we wrote above to get mean RMSE and 2 standard de
viations scores.
en_rmse,en_rmse_2std = negmse_to_rmse(en_negmse_cvscores)

# Printing out the mean RMSE and 2 standard variations for ElasticNet
print('en mean cv RMSE: {:.2f} +- {:.2f}'.format(en_rmse,en_rmse_2std)
)
```

```
en mean cv RMSE: 1.77 +- 0.26
```

In [12]:
```
# Choosing the best model based on mean RMSE,
#     for that, we retrain on the entire training set
#     and report test set RMSE


from sklearn.metrics import mean_squared_error

# Retraining the best performaing model on the entire training set
lr2 = LinearRegression().fit(X_train_r,y_train_r)

# Generating predictions y_pred, again using the training set.
y_pred = lr2.predict(X_train_r)

# Calculating RMSE on the test set using the trained model
test_rmse = np.sqrt(mean_squared_error(y_train_r, y_pred))

print('test RMSE : {:.2f}'.format(test_rmse))
```

```
test RMSE : 1.53
```

Classification

Here we build a model to classify low vs. high adjusted sales price.

Create Classification Target

In [13]:
```
# We'll create a binary target by thresholding at the median of our Ad
jSalePrice
#  High = 1, Low = 0
y_c = (df.AdjSalePrice_x100000 > df.AdjSalePrice_x100000.median()).ast
ype(int)
```

In [14]:
```
# Creating a training and held aside set

X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(X, y_c, te
st_size=0.2, random_state=42)
```

In [15]:
```
# Instead of creating and training a Dummy Classifier,let's
# calculate accuracy if we just predict 1 for all training set items.

rect.
baseline_acc = sum(y_train_c == 1)/ len(y_train_c)

print('baseline accuracy: {:.2f}'.format(baseline_acc))
```

```
baseline accuracy: 0.51
```

In [16]:
```
# training and calculating 5-fold cv accuracy for
#   a LogisticRegression model on the training set



from sklearn.linear_model import LogisticRegression


logr_cvscores = cross_val_score(LogisticRegression(solver='lbfgs'), X_
train_c, y_train_c, cv=5,
                        scoring='accuracy')

# Calculating mean cv accuracy
logr_acc = np.mean(logr_cvscores)

# Calculating 2 standard deviations for the cv scores
logr_acc_2std = 2 * np.std(logr_cvscores)

print('logr mean cv accuracy: {:.2f} +- {:.2f}'.format(logr_acc,logr_a
cc_2std))
```

```
logr mean cv accuracy: 0.75 +- 0.06
```

In [17]:
```
# Performing a 5-fold cross validated grid search over the number of t
rees and tree depth.

from sklearn.ensemble import RandomForestClassifier

params = {'n_estimators':[5,100,200], 'max_depth':[3,5,10]}

gscv = GridSearchCV(RandomForestClassifier(),params,cv=5, refit=True).
fit(X_train_c,y_train_c)

# Printing out the best parameter setting found using grid search and
the best parameter setting found
print('gscv best params: {}'.format(gscv.best_params_))

mean_accuracy = gscv.best_score_
print('rf best accuracy: {:.3f}'.format(mean_accuracy))
```

```
gscv best params: {'max_depth': 10, 'n_estimators': 200}
rf best accuracy: 0.797
```

In [18]:
```
# Evaluating the Random Forest Model on the test set


rf = gscv.best_estimator_

# Calculating accuracy on the test set using the trained model
test_acc = rf.score(X_test_c, y_test_c)
print('test acc : {:.2f}'.format(test_acc))
```

```
test acc : 0.79
```

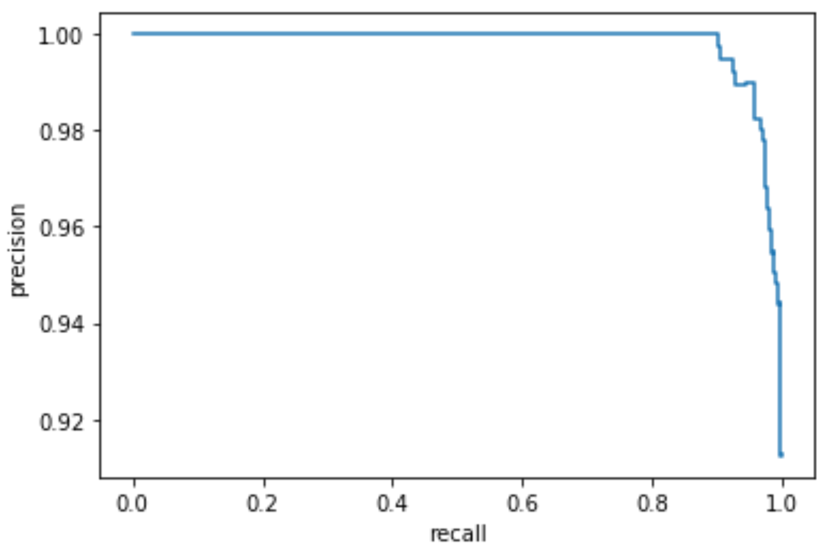
Plotting Precision-Recall curve for the Random Forest model

In [19]:
```
from sklearn.metrics import precision_recall_curve


pypos_rf = rf.predict_proba(X_train_c)

precision, recall, _ = precision_recall_curve(y_train_c, pypos_rf[:,1]
)

plt.step(recall, precision)
plt.xlabel("recall")
plt.ylabel("precision")
plt.show()
```



Feature selection

In [20]:
```
# Using our trained Random Forest model to determine which features ar
e most important for prediction


from sklearn.feature_selection import SelectFromModel


sfm = SelectFromModel(estimator=rf, threshold='mean', prefit=True)

# Getting the selected feature names
kept_columns = list(X.columns[sfm.get_support()])

print('kept columns: {}'.format(kept_columns))
```

```
kept columns: ['SqFtTotLiving_x1000', 'SqFtLot_x1000']
```