10.1 – Machine Learning

## **10.1.4 - Supervised Learning with scikit-learn**

**Chapter-1: Classification**

|  |  |  |
| --- | --- | --- |
| Supervised Learning | | |
| The Iris dataset in scikit-learn |  | In [1]: from sklearn import datasets  In [2]: import pandas as pd  In [3]: import numpy as np  In [4]: import matplotlib.pyplot as plt  In [5]: plt.style.use('ggplot')  In [6]: iris = datasets.load\_iris()  In [7]: type(iris)  Out[7]: sklearn.datasets.base.Bunch  In [8]: print(iris.keys())  dict\_keys(['data', 'target\_names', 'DESCR', 'feature\_names', 'target'])  In [9]: type(iris.data), type(iris.target)  Out[9]: (numpy.ndarray, numpy.ndarray)  In [10]: iris.data.shape  Out[10]: (150, 4)  In [11]: iris.target\_names  Out[11]: array(['setosa', 'versicolor', 'virginica'], dtype='<U10')  In [12]: X = iris.data  In [13]: y = iris.target  In [14]: df = pd.DataFrame(X, columns=iris.feature\_names)  In [15]: print(df.head())  In [16]: \_ = pd.scatter\_matrix(df, c = y, figsize = [8, 8], s=150, marker = 'D') |
| Visual EDA  (Exercise) | Tek bir degiskenin, baska bir degisken ile gruplandirilarak gosterilmesi | plt.figure()  sns.countplot(x='education', hue='party', data=df, palette='RdBu')  plt.xticks([0,1], ['No', 'Yes'])  plt.show() |
| The classification challenge | | |
| k-Nearest Neighbors  Using scikit-learn to fit a classifier | **.fit() method**  Not:  1# Missing value olmamali  2# target’daki satir sayisi da ayni olmali | In [1]: from sklearn.neighbors import KNeighborsClassifier  In [2]: knn = KNeighborsClassifier(n\_neighbors=6)  In [3]: knn.fit(iris['data'], iris['target'])  Out[3]: KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',metric\_params=None, n\_jobs=1, n\_neighbors=6, p=2,weights='uniform')  In [4]: iris['data'].shape  Out[4]: (150, 4)  In [5]: iris['target'].shape  Out[5]: (150,) |
| Predicting on unlabeled data | **.predict() method**  Not:  1# predict edilecek olan data’daki sutun sayisi, predictor sayisi ile ayni olmali | In [6]: prediction = knn.predict(X\_new)  In [7]: X\_new.shape  Out[7]: (3, 4)  In [8]: print('Prediction {}’.format(prediction))  Prediction: [1 1 0] |
| Exercise:  k-Nearest Neighbors: Fit | Not:  Tutorialdan farkli olarak, “.fit() method’a gececegi features array’leri onceden olusturmus | from sklearn.neighbors import KNeighborsClassifier  # Create arrays for the features and the response variable  y = df['party'].values  X = df.drop('party', axis=1).values  # Create a k-NN classifier with 6 neighbors  knn = KNeighborsClassifier(n\_neighbors=6)  # Fit the classifier to the data  knn.fit(X, y)  # Predict the labels for the training data X  y\_pred = knn.predict(X)  new\_prediction = knn.predict(X\_new)  print("Prediction: {}".format(new\_prediction)) |
| Measuring model performance | | |
| Train/test split |  | In [1]: from sklearn.model\_selection import train\_test\_split  In [2]: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=21, stratify=y)  In [3]: knn = KNeighborsClassifier(n\_neighbors=8)  In [4]: knn.fit(X\_train, y\_train)  In [5]: y\_pred = knn.predict(X\_test)  In [6]: print("Test set predictions:\n {}".format(y\_pred))  Test set predictions:  [2 1 2 2 1 0 1 0 0 1 0 2 0 2 2 0 0 0 1 0 2 2 2 0 1 1 1 0 0  1 2 2 0 0 2 2 1 1 2 1 1 0 2 1]  In [7]: knn.score(X\_test, y\_test)  Out[7]: 0.9555555555555556 |
| The digits recognition dataset | Not:  Her bir data satiri, bir “image”ait 64 pixel’in herbirindeki siyahlik oranini ifade eden 1-16 arasindaki rakamlardan olusmus. Bir anlamda 2 boyutluluk tek boyuta indirgenmis. Yani her bir pixel bir feature haline gelmis (64 feature for each image). Feature numaralandirmasi sol usten ve soldan saga olacak sekilde yapilmis. | from sklearn import datasets  import matplotlib.pyplot as plt  digits = datasets.load\_digits()  print(digits.keys())  print(digits.DESCR)  print(digits.images.shape)  print(digits.data.shape)  # Display digit 1010  plt.imshow(digits.images[1010], cmap=plt.cm.gray\_r, interpolation='nearest')  plt.show() |
| Exercise:  Overfitting and underfitting | Not:  Hangi “k” sayisinin daha iyi sonuc verecegini bulmak icin yazilmis bir “for loop”  \*np.arange: -1 adet elemanli list yapar (int64)  \*npempty: girilen uzunluk adedince tum elemanli “0” olan list yapar (float64)  \*enumerate(): mevcut listenin her bir elemanin “0”dan baslayan sayilari ekler. Boylece ikiser elemanli tuple’lardan olusan bir liste elde edilmis olur. Iterasyon islemleri icin kullanilir | # Setup arrays to store train and test accuracies  neighbors = np.arange(1, 9)  train\_accuracy = np.empty(len(neighbors))  test\_accuracy = np.empty(len(neighbors))  # Loop over different values of k  for i, k in enumerate(neighbors):  knn = KNeighborsClassifier(n\_neighbors=k)  knn.fit(X\_train, y\_train)  train\_accuracy[i] = knn.score(X\_train, y\_train)  test\_accuracy[i] = knn.score(X\_test, y\_test)  # Generate plot  plt.title('k-NN: Varying Number of Neighbors')  plt.plot(neighbors, test\_accuracy, label = 'Testing Accuracy')  plt.plot(neighbors, train\_accuracy, label = 'Training Accuracy')  plt.legend()  plt.xlabel('Number of Neighbors')  plt.ylabel('Accuracy')  plt.show() |

Chapter-2: Regression

|  |  |  |
| --- | --- | --- |
| Introduction to regression | | |
| Predicting house value from a single feature | Not:  1# Buradaki, “.reshape()” girilen (satir, sutun) sayisina gore data’yi yeniden sekillendiriyor. “-1”in islevi, “unknown” olarak tanitma. Sadece bir tanesi “-1” olabilir. Anlami su, degeri “-1” olmayan parametreyi kac girdiysem o kadar yap, “-1” olan parametrenin sayisini da ona gore ayarla, artik kaca tekabul ediyorsa | In [1]: boston = pd.read\_csv('boston.csv')  In [3]: X = boston.drop('MEDV', axis=1).values  In [4]: y = boston['MEDV'].values  In [5]: X\_rooms = X[:,5]  In [6]: type(X\_rooms), type(y)  Out[6]: (numpy.ndarray, numpy.ndarray)  In [7]: y = y.reshape(-1, 1)  In [8]: X\_rooms = X\_rooms.reshape(-1, 1)  In [9]: plt.scatter(X\_rooms, y)  In [10]: plt.ylabel('Value of house /1000 ($)')  In [11]: plt.xlabel('Number of rooms')  In [12]: plt.show(); |
|  | Fitting a regression model  Not:  2# linspace method, default olarak 50 deger uretiyor. | In [13]: import numpy as np  In [14]: from sklearn import linear\_model  In [15]: reg = linear\_model.LinearRegression()  In [16]: reg.fit(X\_rooms, y)  In [17]: prediction\_space = np.linspace(min(X\_rooms), max(X\_rooms)).reshape(-1, 1)  In [18]: plt.scatter(X\_rooms, y, color='blue')  In [19]: plt.plot(prediction\_space, reg.predict(prediction\_space), color='black', linewidth=3)  In [20]: plt.show() |
| heatmap |  | sns.heatmap(df.corr(), square=True, cmap='RdYlGn') |
| The basics of linear regression | | |
|  | Regression mechanics | ● y = ax + b  ● y = target  ● x = single feature  ● a, b = parameters of model  Ordinary least squares (OLS)  "y = a1x1 + a2x2 + a3x3 + anxn + b" |
| Linear regression on all features |  | In [1]: from sklearn.model\_selection import train\_test\_split  In [2]: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state=42)  In [3]: reg\_all = linear\_model.LinearRegression()  In [4]: reg\_all.fit(X\_train, y\_train)  In [5]: y\_pred = reg\_all.predict(X\_test)  In [6]: reg\_all.score(X\_test, y\_test)  Out[6]: 0.71122600574849526 |
| Train/test split for regression |  | from sklearn.linear\_model import LinearRegression  from sklearn.metrics import mean\_squared\_error  from sklearn.model\_selection import train\_test\_split  # Create training and test sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state=42)  # Create the regressor: reg\_all  reg\_all = LinearRegression()  # Fit the regressor to the training data  reg\_all.fit(X\_train, y\_train)  # Predict on the test data: y\_pred  y\_pred = reg\_all.predict(X\_test)  # Compute and print R^2 and RMSE  print("R^2: {}".format(reg\_all.score(X\_test, y\_test)))  rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))  print("Root Mean Squared Error: {}".format(rmse)) |
| Cross-validation | | |
| K-Fold CV |  | In [1]: from sklearn.model\_selection import cross\_val\_score  In [2]: reg = linear\_model.LinearRegression()  In [3]: cv\_results = cross\_val\_score(reg, X, y, cv=5)  In [4]: print(cv\_results)  [ 0.63919994 0.71386698 0.58702344 0.07923081 -0.25294154]  In [5]: np.mean(cv\_results)  Out[5]: 0.35327592439587058 |
| Regularized regression | | |
| Why regularize? |  | ● Recall: Linear regression minimizes a loss function  ● It chooses a coefficient for each feature variable  ● Large coefficients can lead to overfi"ing  ● Penalizing large coefficients: Regularization |
| Ridge regression | when building regression models, Ridge regression should be your first choice. | ●  ● Alpha: Parameter we need to choose  ● Picking alpha here is similar to picking k in k-NN  ● Hyperparameter tuning (More in Chapter 3)  ● Alpha controls model complexity  ● Alpha = 0: We get back OLS (Can lead to overfi"ing)  ● Very high alpha: Can lead to underftting |
| Ridge regression in scikit-learn | Not:  Alpha katsayisi kadar oynamalar yapar regresyon cizgisinde. Cati aynidir. | In [1]: from sklearn.linear\_model import Ridge  In [2]: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state=42)  In [3]: ridge = Ridge(alpha=0.1, normalize=True)  In [4]: ridge.fit(X\_train, y\_train)  In [5]: ridge\_pred = ridge.predict(X\_test)  In [6]: ridge.score(X\_test, y\_test)  Out[6]: 0.69969382751273179 |
| Lasso regression in scikit-learn | Not:  Burada da outlier’lari elimine edilmesi on plana cikiyor | Lasso is great for feature selection  In [1]: from sklearn.linear\_model import Lasso  In [2]: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state=42)  In [3]: lasso = Lasso(alpha=0.1, normalize=True)  In [4]: lasso.fit(X\_train, y\_train)  In [5]: lasso\_pred = lasso.predict(X\_test)  In [6]: lasso.score(X\_test, y\_test)  Out[6]: 0.59502295353285506 |
| Lasso for feature selection in scikit-learn |  | In [1]: from sklearn.linear\_model import Lasso  In [2]: names = boston.drop('MEDV', axis=1).columns  In [3]: lasso = Lasso(alpha=0.1)  In [4]: lasso\_coef = lasso.fit(X, y).coef\_  In [5]: \_ = plt.plot(range(len(names)), lasso\_coef)  In [6]: \_ = plt.xticks(range(len(names)), names, rotation=60)  In [7]: \_ = plt.ylabel('Coefficients')  In [8]: plt.show() |
|  | Not:  logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None)[source])  girilen degerler arasinda ve girilen deger adedince 10 tabaninda logaritmik degerler uretir |  |

Chapet-3: Fine-tuning your model

|  |  |  |
| --- | --- | --- |
| How good is your model? | | |
| Diagnosing classification predictions |  |  |
| Metrics from the confusion matrix | **Precision=**1/positive dediklerinden gercekten positive olanlarin orani  (1’diye tahmin ettigin 100 hastanin 70 tanesinin 1 olmasi, 30 tanesinin 0 olmasi gibi.  **Recall=True positive rate = Sensitivity** (1’leri/positive’leri tutturma orani) – 100 tane 1’den 80 tanesini dogru tahmin etmek gibi  **Specificity** = 0’lari/negative’leri tutturma orani - 100 tane 0’dan 80 tanesini dogru tahmin etmek gibi  **False positive rate= 1-Specificity** |  |
| Confusion matrix in scikit-learn |  | …  In [6]: y\_pred = knn.predict(X\_test)  In [7]: print(confusion\_matrix(y\_test, y\_pred))  In [8]: print(classification\_report(y\_test, y\_pred)) |
| Logistic regression and the ROC curve | | |
| Logistic regression in scikit-learn |  | In [1]: from sklearn.linear\_model import LogisticRegression  In [2]: from sklearn.model\_selection import train\_test\_split  In [3]: logreg = LogisticRegression()  In [4]: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,  ...: test\_size=0.4, random\_state=42)  In [5]: logreg.fit(X\_train, y\_train)  In [6]: y\_pred = logreg.predict(X\_test) |
| Plotting the ROC curve |  | In [1]: from sklearn.metrics import roc\_curve  In [2]: y\_pred\_prob = logreg.predict\_proba(X\_test)[:,1]  In [3]: fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_prob)  In [4]: plt.plot([0, 1], [0, 1], 'k--')  In [5]: plt.plot(fpr, tpr, label='Logistic Regression')  In [6]: plt.xlabel('False Positive Rate’)  In [7]: plt.ylabel('True Positive Rate')  In [8]: plt.title('Logistic Regression ROC Curve')  In [9]: plt.show(); |
|  | **Y ekseni = tpr, X ekseni fpr iken her iki degerin 1 alabildigi bir grafik nasil elde edebiliriz**  **P=1 iken her ikisi de 0 demek de nedir veya p=0 iken her ikisi 1 ne demektir**  **Cok kafami karistirdi bu denklem** |  |
| Area under the ROC curve |  |  |
|  |  | Larger area under the ROC curve = beter model |
|  |  | In [1]: from sklearn.metrics import roc\_auc\_score  In [2]: logreg = LogisticRegression()  In [3]: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=42)  In [4]: logreg.fit(X\_train, y\_train)  In [5]: y\_pred\_prob = logreg.predict\_proba(X\_test)[:,1]  In [6]: roc\_auc\_score(y\_test, y\_pred\_prob)  Out[6]: 0.997466216216 |
| AUC using cross-validation |  | In [7]: from sklearn.model\_selection import cross\_val\_score  In [8]: cv\_scores = cross\_val\_score(logreg, X, y, cv=5, scoring='roc\_auc')  In [9]: print(cv\_scores)  [ 0.99673203 0.99183007 0.99583796 1. 0.96140652] |
| Hyperparameter  tuning |  | ● Linear regression: Choosing parameters  ● Ridge/lasso regression: Choosing alpha  ● k-Nearest Neighbors: Choosing n\_neighbors  ● Parameters like alpha and k: Hyperparameters  ● Hyperparameters cannot be learned by fitting the model |
| GridSearchCV in scikit-learn |  | In [1]: from sklearn.model\_selection import GridSearchCV  In [2]: param\_grid = {'n\_neighbors': np.arange(1, 50)}  In [3]: knn = KNeighborsClassifier()  In [4]: knn\_cv = GridSearchCV(knn, param\_grid, cv=5)  In [5]: knn\_cv.fit(X, y)  In [6]: knn\_cv.best\_params\_  Out[6]: {'n\_neighbors': 12}  In [7]: knn\_cv.best\_score\_  Out[7]: 0.933216168717 |
| Exercise:  Hyperparameter tuning with GridSearchCV | logistic regression also has a regularization parameter: C  . C controls the inverse of the regularization strength, and this is what you will tune in this exercise. A large C can lead to an overfit model, while a small C can lead to an underfit model. | from sklearn.linear\_model import LogisticRegression  from sklearn.model\_selection import GridSearchCV  # Setup the hyperparameter grid  c\_space = np.logspace(-5, 8, 15)  param\_grid = {'C': c\_space}  # Instantiate a logistic regression classifier: logreg  logreg = LogisticRegression()  # Instantiate the GridSearchCV object: logreg\_cv  logreg\_cv = GridSearchCV(logreg, param\_grid, cv=5)  # Fit it to the data  logreg\_cv.fit(X, y)  # Print the tuned parameters and score  print("Tuned Logistic Regression Parameters: {}".format(logreg\_cv.best\_params\_))  print("Best score is {}".format(logreg\_cv.best\_score\_)) |
| Hyperparameter tuning with RandomizedSearchCV |  | # Import necessary modules  from scipy.stats import randint  from sklearn.tree import DecisionTreeClassifier  from sklearn.model\_selection import RandomizedSearchCV  # Setup the parameters and distributions to sample from: param\_dist  param\_dist = {"max\_depth": [3, None],  "max\_features": randint(1, 9),  "min\_samples\_leaf": randint(1, 9),  "criterion": ["gini", "entropy"]}  # Instantiate a Decision Tree classifier: tree  tree = DecisionTreeClassifier()  # Instantiate the RandomizedSearchCV object: tree\_cv  tree\_cv = RandomizedSearchCV(tree, param\_dist, cv=5)  # Fit it to the data  tree\_cv.fit(X, y)  # Print the tuned parameters and score  print("Tuned Decision Tree Parameters: {}".format(tree\_cv.best\_params\_))  print("Best score is {}".format(tree\_cv.best\_score\_)) |
| Exercise:  Hold-out set in practice II: Regression |  | # Import necessary modules  from sklearn.linear\_model import ElasticNet  from sklearn.metrics import mean\_squared\_error  from sklearn.model\_selection import GridSearchCV, train\_test\_split  # Create train and test sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=42)  # Create the hyperparameter grid  l1\_space = np.linspace(0, 1, 30)  param\_grid = {'l1\_ratio': l1\_space}  # Instantiate the ElasticNet regressor: elastic\_net  elastic\_net = ElasticNet()  # Setup the GridSearchCV object: gm\_cv  gm\_cv = GridSearchCV(elastic\_net, param\_grid, cv=5)  # Fit it to the training data  gm\_cv.fit(X\_train, y\_train)  # Predict on the test set and compute metrics  y\_pred = gm\_cv.predict(X\_test)  r2 = gm\_cv.score(X\_test, y\_test)  mse = mean\_squared\_error(y\_test, y\_pred)  print("Tuned ElasticNet l1 ratio: {}".format(gm\_cv.best\_params\_))  print("Tuned ElasticNet R squared: {}".format(r2))  print("Tuned ElasticNet MSE: {}".format(mse)) |

Chapter-4: Preprocessing and pipelines

|  |  |  |
| --- | --- | --- |
| Preprocessing data |  |  |
| Dealing with categorical features | ● scikit-learn: OneHotEncoder()  ● pandas: get\_dummies() | ● Scikit-learn will not accept categorical features by default  ● Need to encode categorical features numerically |
| Encoding dummy variables |  | In [1]: import pandas as pd  In [2]: df = pd.read\_csv('auto.csv')  In [3]: df\_origin = pd.get\_dummies(df)  In [4]: print(df\_origin.head())  In [5]: df\_origin = df\_origin.drop('origin\_Asia', axis=1) |
|  | Command chain | In [10]: ridge = Ridge(alpha=0.5, normalize=True).fit(X\_train, y\_train) |
|  |  | # Create dummy variables with drop\_first=True: df\_region  df\_region = pd.get\_dummies(df, drop\_first=True) |
| Handling missing data | | |
|  | **.replace( )** | In [8]: df.insulin.replace(0, np.nan, inplace=True)  In [9]: df.triceps.replace(0, np.nan, inplace=True)  In [10]: df.bmi.replace(0, np.nan, inplace=True)  In [11]: df.info() |
| 1st way: Dropping missing data | **.dropna( )** | In [12]: df = df.dropna()  In [13]: df.shape |
| 2nd way: Imputing missing data |  | In [1]: from sklearn.preprocessing import Imputer  In [2]: imp = Imputer(missing\_values='NaN', strategy='mean', axis=0)  In [3]: imp.fit(X)  In [4]: X = imp.transform(X) |
| 3rd way: Imputing within a pipeline |  | In [1]: from sklearn.pipeline import Pipeline  In [2]: from sklearn.preprocessing import Imputer  In [3]: imp = Imputer(missing\_values='NaN', strategy='mean', axis=0)  In [4]: logreg = LogisticRegression()  In [5]: steps = [('imputation', imp), ('logistic\_regression', logreg)]  In [6]: pipeline = Pipeline(steps)  In [7]: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  In [8]: pipeline.fit(X\_train, y\_train)  In [9]: y\_pred = pipeline.predict(X\_test)  In [10]: pipeline.score(X\_test, y\_test)  Out[10]: 0.75324675324675328 |
| eXERCISE |  |  |
| Imputing missing data in a ML Pipeline I |  | from sklearn.preprocessing import Imputer  from sklearn.svm import SVC  # Setup the Imputation transformer: imp  imp = Imputer(missing\_values='NaN', strategy='most\_frequent', axis=0)  # Instantiate the SVC classifier: clf  clf = SVC()  # Setup the pipeline with the required steps: steps  steps = [('imputation', imp),  ('SVM', clf)] |
| Centering and Scaling | | |
| Ways to normalize your data |  | ● Standardization: Subtract the mean and divide by variance  ● All features are centered around zero and have variance one  ● Can also subtract the minimum and divide by the range  ● Minimum zero and maximum one  ● Can also normalize so the data ranges from -1 to +1  ● See scikit-learn docs for further details |
| Scaling in scikit-learn |  | In [2]: from sklearn.preprocessing import scale  In [3]: X\_scaled = scale(X)  In [4]: np.mean(X), np.std(X)  Out[4]: (8.13421922452, 16.7265339794)  In [5]: np.mean(X\_scaled), np.std(X\_scaled)  Out[5]: (2.54662653149e-15, 1.0) |
| Scaling in a pipeline |  | In [6]: from sklearn.preprocessing import StandardScaler  In [7]: steps = [('scaler', StandardScaler()),  ...: ('knn', KNeighborsClassifier())]  In [8]: pipeline = Pipeline(steps)  In [9]: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,  ...: test\_size=0.2, random\_state=21)  In [10]: knn\_scaled = pipeline.fit(X\_train, y\_train)  In [11]: y\_pred = pipeline.predict(X\_test)  In [12]: accuracy\_score(y\_test, y\_pred)  Out[12]: 0.956  In [13]: knn\_unscaled = KNeighborsClassifier().fit(X\_train, y\_train)  In [14]: knn\_unscaled.score(X\_test, y\_test)  Out[14]: 0.928 |
| CV and scaling in a pipeline |  | In [14]: steps = [('scaler', StandardScaler()),  ...: (('knn', KNeighborsClassifier())]  In [15]: pipeline = Pipeline(steps)  In [16]: parameters = {knn\_\_n\_neighbors=np.arange(1, 50)}  In [17]: X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y,  ...: test\_size=0.2, random\_state=21)  In [18]: cv = GridSearchCV(pipeline, param\_grid=parameters)  In [19]: cv.fit(X\_train, y\_train)  In [20]: y\_pred = cv.predict(X\_test)  In [21]: print(cv.best\_params\_){'knn\_\_n\_neighbors': 41}  In [22]: print(cv.score(X\_test, y\_test))  0.956  In [23]: print(classification\_report(y\_test, y\_pred)) |
| Exercise |  | from sklearn.preprocessing import StandardScaler  from sklearn.pipeline import Pipeline  # Setup the pipeline steps: steps  steps = [('scaler', StandardScaler()),  ('knn', KNeighborsClassifier())]  # Create the pipeline: pipeline  pipeline = Pipeline(steps)  # Create train and test sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)  # Fit the pipeline to the training set: knn\_scaled  knn\_scaled = pipeline.fit(X\_train, y\_train)  # Instantiate and fit a k-NN classifier to the unscaled data  knn\_unscaled = KNeighborsClassifier().fit(X\_train, y\_train)  # Compute and print metrics  print('Accuracy with Scaling: {}'.format(knn\_scaled.score(X\_test, y\_test)))  print('Accuracy without Scaling: {}'.format(knn\_unscaled.score(X\_test, y\_test))) |
| Bringing it all together I: Pipeline for classification |  | # Setup the pipeline  steps = [('scaler', StandardScaler()),  ('SVM', SVC())]  pipeline = Pipeline(steps)  # Specify the hyperparameter space  parameters = {'SVM\_\_C':[1, 10, 100],  'SVM\_\_gamma':[0.1, 0.01]}  # Create train and test sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=21)  # Instantiate the GridSearchCV object: cv  cv = GridSearchCV(pipeline, param\_grid=parameters)  # Fit to the training set  cv.fit(X\_train, y\_train)  # Predict the labels of the test set: y\_pred  y\_pred = cv.predict(X\_test)  # Compute and print metrics  print("Accuracy: {}".format(cv.score(X\_test, y\_test)))  print(classification\_report(y\_test, y\_pred))  print("Tuned Model Parameters: {}".format(cv.best\_params\_)) |

Mini Project-1

|  |  |  |
| --- | --- | --- |
| Fitting Linear Regression using statsmodels | | |
| Scatter Plot |  | plt.scatter(bos.B, bos.PRICE)  plt.xlabel("the proportion of blacks by town (B)")  plt.ylabel("Housing Price")  plt.title("Relationship between B and Price") |
| Seaborn regplot |  | sns.regplot(y="PRICE", x="B", data=bos, fit\_reg = True) |
| Histogram |  | plt.hist(np.log(bos.CRIM))  plt.title("CRIM")  plt.xlabel("Crime rate per capita")  plt.ylabel("Frequencey")  plt.show() |
| OLS fit |  | import statsmodels.api as sm  from statsmodels.formula.api import ols  m = ols('PRICE ~ RM',bos).fit()  print(m.summary())  In general, a formula for an OLS multiple linear regression is  Y ~ X1 + X2 + ... + Xp |
| Fitting Linear Regression using sklearn |  | from sklearn.linear\_model import LinearRegression  X = bos.drop('PRICE', axis = 1)  # This creates a LinearRegression object  lm = LinearRegression() |
| What can you do with a LinearRegression object? |  | Main functions Description  **lm.fit()** 🡪 Fit a linear model  **lm.predit()** 🡪 Predict Y using the linear model with estimated coefficients  **lm.score()** 🡪 Returns the coefficient of determination (R^2). A measure of how well observed outcomes are replicated by the model, as the proportion of total variation of outcomes explained by the model |
| What output can you get? |  | Output Description  **lm.coef\_** 🡪Estimated coefficients  **lm.intercept\_** 🡪Estimated intercept |
|  |  |  |
|  |  |  |

<https://thegyre.io/coursework/ml/linear_regression/>