Programming-5

28 July 2022 21:40

Load	`from sklearn.datasets import fetch_openml`
	
	`X,y = fetch_openml('mnist_784', version=1, return_X_y = True)`

Error	`Getting the index 2022 in X`
	`X = dataframe`
	
	`X.to_numpy or np.asarray(X) or X.iloc[2022]`

Model	`6 as positive class and 9 as negative class`
	
	`y_train_6 = 1 *  np.ones(len(y_train[y_train=='6']))`
	
	`Combine 1s and -1s`
	
	`y_train_combine = np.concatenate ( (y_train_6,y_train_9 ),axis=0)`
	`Notice the axis`

Shuffle	from sklearn.utils import shuffle
Pereceptron	Skelarn.linear_model import Perceptron learning rate = eta0 epochs = max_iter Include the intercept (bias) term = fit_intercept=True
w weights assigend for the 69th feature after 10 iterations	`perceptron.coef_[:,69]`

sequence of bias (intercept)	`for i in range(1,6):`
	`  perceptron = Perceptron(eta0=`
	`1, max_iter = i, shuffle=False, random_state=`
	`1729, fit_intercept=True)`
	`  perceptron.fit(X_train_combine,y_train_combine)`
	`  print(perceptron.intercept_)`
	
	`Change max_iter to i, and keep i from 1`

Graded

Confusion matrix display	`from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay`
	`cm = confusion_matrix(y_test_combine,y_pred)`
	
	`disp = ConfusionMatrixDisplay(cm)`
	`disp.plot()`

Applying Linear PCA	`from sklearn.decomposition import PCA`
	
	`pca = PCA( n_components=10, random_state=1)`

```
X_train_reduced = pca.fit(X_train_combine).transform(X_train_combine)
X_test_reduced = pca.fit(X_test_combine).transform(X_test_combine)

Notice the method: first fit then transform

Perceptron = Perceptron.fit(X_train_reduced,y_train)

Y_predict = Perceptron.predict(X_train_reduced,y_train)
```

Programming-6

29 July 2022 03:50

Splitting data	`X_train,X_test, y_train,y_test = X[:49000],X[49000:],y[:49000],y[49000:]` Actually In **train_test_split** you have parameter named shuffle which helps you to **shuffle** the dataset. If you set it to False it works similar to slicing only Change shuffle=False
Shuffe the datamatrix	`from sklearn.utils import shuffle` `X_train_69,y_train_69=shuffle(X_train_69,y_train_69,random_state=1729)`

logistic regression with SGDclassifier -> LOSS	`max_iter=1`
	
	`from sklearn.metrics import log_loss`
	
	`clfA = SGDClassifier(loss='log', random_state=10, max_iter=1, alpha=0.01, warm_start=True)`
	
	`loss =[]`
	`iterations = 10`
	`for i in range (iterations):`
	`  clfA.fit(X_train_69,y_train_69)`
	`  y_pred = clfA.predict_proba(X_train_69)`
	`  loss.append(log_loss(y_train_69,y_pred))`
	`plt.plot(loss)`

SGDClassifier	Learning rate = eta0 when learning rate is given keep eta0 = given value and **Learning_rate = 'constant'** No regularisation keep alpha=0

First False positive Index	`FPidx = []`
	`for i in range(len(y_train_69)):`
	`  if y_pred3[i]==1 and y_train_69[i]==0:`
	`  FPidx.append(i)`
	`FPidx[0]`

Load Data	`from sklearn.datasets import fetch_20newsgroups`
	
	`X,y = fetch_20newsgroups(return_X_y=True)`

MultinomialNB	`from sklearn.naive_bayes import MultinomialNB`
	`from sklearn.feature_extraction.text import TfidfVectorizer`
	
	`from sklearn.pipeline import Pipeline`
	`nb = Pipeline([('tfid', TfidfVectorizer()),('mnb',Multin`

	omialNB())]) nb.fit(X_train,y_train) nb.score(X_test,y_test)

Programming-7

30 July 2022 03:48

Count vectorizer	`from sklearn.feature_extraction.text import CountVectorizer`
	`cvc = CountVectorizer()`
	`mat = Cvc.fit_transform(text_data)`
	
	Converting the countvectorizer object in array
	
	`cvc_arr = cvc.toarray()`
	
	column wise sum to know the count of appearances of object in the whole document
	
	`np.sum(np.where(np.sum(cvc_arr,axis=0)>=2,1,0))`
	
	sum of all the words which appear more than 2 times in document

Token associated with a certain word	`cvc = CountVectorizer()`
	
	`cvc.vocabulary_`

Sparse to dense output	`print(mat.todense())`
	
	`[[0 0 1 1 0 1 0 1 0 1]`
	`[0 2 0 1 0 0 1 1 0 1]`
	`[1 0 0 1 1 0 0 1 1 1]]`

Closest vector to given vecotrs	`X = matrix`
	
	`from sklearn.neighbors import NearestNeighbors`
	`nn = NearestNeighbors(n_neighbors=k)`
	`nn.fit(X)`
	`closest_vec = X[nn.kneighbors(np.atleast_2d(p))[1][0,0]]`
	`print(closest_vec)`
	
	OR
	
	`nn.kneighbors(p)`

Graded

Load	`from sklearn.datasets import load_digits`

Reshaping the dataset	`X.shape,y.shape`
	
	`((90000, 10), (90000,))`
	Reshape the dataset in such a way that each entry of data has 90 samples
	
	`[27] X_ = X_array.reshape(-1,90,10)`
	`y_ = y_array.reshape(-1,90,1)`

Partial fit	`#selecting data in batches` `for i in range(X_train.shape[0]):`

	`  X_batch, y_batch = X_train[i],y_train[i]`
	`  #partially fitting data in batches`
	`  regressor.partial_fit(X_batch,y_batch)`

R2_Score, mse score	`y_preds = []`
	`for i in range(X_test.shape[0]):`
	`  y_pred = regressor.predict(X_test[i])`
	`  y_preds.append(y_pred)`
	
	`Reshape the arrays as single dimension`
	
	`y_test_1 = y_test.reshape(-1)`
	`y_pred_1 = np.asarray(y_preds).reshape(-1)`
	
	`r2 = r2_score(y_test_1,y_pred_1)`

Normalizer	`from sklearn.preprocessing import Normalizer`
	
	`scaler = Normalizer()`
	`scaler.fit(X_train)`
	
	`X_train_trans =  scaler.transform(X_train)`
	`X_test_trans = scaler.transform(X_test)`

Accuracy with KNN classifier	`from sklearn.metrics import accuracy_score`
	`for i in range (2,5):`
	`  knn = KNeighborsClassifier(i)`
	`  knn.fit(X_train_trans,y_train)`
	`  y_pred = knn.predict(X_test_trans)`
	`  print(f'For k={i}, accuracy is {(accuracy_score(y_test,y_pred)):.3f}')`

Programming-8

30 July 2022 17:11

Model	`from sklearn.svm import SVC`
	
	`svc.fit(X_train,y_train)`
	`Score = svc.score(X_test,y_test)`

Import the iris dataset and drop the rows where class=Iris-versicolor.	`df = load_iris()`
	`print(df.DESCR)`
	
	`Setosa is class 1`
	
	`X2 = X[y!=1]`
	`y2 = y[y!=1]`

sum of the main diagonal elements of the confusion matrix	`con_matrix = confusion_matrix(y_test, svc_pipe.predict (X_test))` `con_matrix.diagonal().sum()`

Programming-9

16 July 2022 20:48

Decision Tree Classifier

Find Depth = The depth of a tree is the maximum distance between the root and any leaf.
.get_depth() OR **.tree_.max_depth**
find Total leaves - **.get_n_leaves()**
minimum samples required to split - **min_samples_split**
minimum impurity decrease - **min_impurity_decrease**
Entropy at the root node - Visualize the graph
minimum number of samples per leaf - **min_samples_leaf**

Graded Assignment
max number of features = **max_features**=
cost complexity pruning parameter - **ccp_alpha** =
minimum number of samples per leaf - **min_samples_leaf=**
Hyperparameter tuning
GridSearchCV (model_selection)
Param_grid
Cv
return_train_score

Cv_search.**score**(X_test,y_test)

MNIST-10

17 July 2022 20:12

Lectures

Load	Keras.datasets import mnist
Model building	Model_selection.**ShuffleSplit** **metrics**.classificaiton_report f' string {variable:**.3f** } to round off
Evaluation	Metrics - classificaiton_report confusion_matrix ConfusionMatrixDisplay Steps, 1. cm=confusion_matrix(y_test,y_pred) 2. disp= ConfusionMatrixDisplay(confusion_matrix=cm) 3. **disp.plot()**

BaggingClassifier

Load	.ensemble.BaggingClassifier
Model	Cross validation (model_selection) cross_val_score score.mean() and score.std() gives only test set score

California housing-10

18 July 2022 00:06

Decision Tree Regressor

Cross_validate	return_score=True return_estimator=True Cv_results[**train_score**] Cv_results[**test_score**]
Cross validation	search_cv = RandomizedSearchCV (model_selection) param_distributions= .cv_results_ cv_results_[**mean_test_score**] cv_results_[**std_test_score**]
Scoring	-search_cv.score(X_test, y_test) notice the negative sign

MNIST -boosting-10

18 July 2022 20:28

Load	ensemble.AdaBoostClassifier ensemble.GradientBoostingClassifier xgboost.XGBClassifier

California -boosting-10

19 July 2022 01:13

Load	xgboost.XGBRegressor

Programming-11

20 July 2022 22:06

LOAD	Sklearn.cluster.Kmeans digit = sklearn.datasets.load_digit digit will be a bunch object … digit.data will contain data

```
pipeline = Pipeline([('scaler',MinMaxScaler()),
                     ('kmeans',KMeans(n_clusters=10, **kmeans_kwargs))])
pipeline.fit(digits.data)
```

Finding optimal number of classes	Elbow Method
	
	`kmeans.inertia_`
	
	Silhouette Method
	
	`from sklearn.metrics import silhouette_score`
	
	`Score = silhouette_score(digits.data, kmeans.labels_)`
	
	`Where kmeans = KMeans(n_clusters=k, **kmeans_kwargs)`

Practice assignment

Load	Iris dataset = **load_iris**

Agglomerative clustering

Load	Sklearn.cluster -> AgglomerativeClustering

Graded Assignment

Drop missing values - **specify axis**
Unique values - **.value_counts()**
Kmeans(X,y)