Preprocessing and Pipelines

Contents

Learn how to impute missing values, convert categorical data to numeric values, scale data, evaluate multiple supervised learning models simultaneously, and build pipelines to streamline your workflow!

Preprocessing data

Creating dummy variables

Being able to include categorical features in the model building process can enhance performance as they may add information that contributes to prediction accuracy.

The `music_df` dataset has been preloaded for you, and its shape is printed. Also, `pandas` has been imported as `pd`.

Now you will create a new DataFrame containing the original columns of `music_df` plus dummy variables from the `“genre”` column.

- Use a relevant function, passing the entire `music_df` DataFrame, to create `music_dummies`, dropping the first binary column.
- Print the shape of `music_dummies`.

```
# edited/added
music_df = pd.read_csv("archive/Supervised-Learning-with-scikit-
learn/datasets/music_clean.csv", index_col=[0])

# Create music_dummies
music_dummies = pd.get_dummies(music_df, drop_first=True)

# Print the new DataFrame's shape
print("Shape of music_dummies: {}".format(music_dummies.shape))
```

```
## Shape of music_dummies: (1000, 12)
```

As there were ten values in the `“genre”` column, nine new columns were added by a call of `pd.get_dummies()` using `drop_first=True`. After dropping the original `“genre”` column, there are still eight new columns in the DataFrame!

Regression with categorical features

Now you have created `music_dummies`, containing binary features for each song’s genre, it’s time to build a ridge regression model to predict song popularity.

`music_dummies` has been preloaded for you, along with `Ridge`, `cross_val_score`, `numpy` as `np`, and a `KFold` object stored as `kf`.

The model will be evaluated by calculating the average RMSE, but first, you will need to convert the scores for each fold to positive values and take their square root. This metric shows the average error of our model's predictions, so it can be compared against the standard deviation of the target value—"`popularity`".

- Create `X`, containing all features in `music_dummies`, and `y`, consisting of the "`popularity`" column, respectively.
- Instantiate a ridge regression model, setting `alpha` equal to 0.2.
- Perform cross-validation on `X` and `y` using the ridge model, setting `cv` equal to `kf`, and using negative mean squared error as the scoring metric.
- Print the RMSE values by converting negative `scores` to positive and taking the square root.

```
# Create X and y
X = music_dummies.drop("popularity", axis=1).values
y = music_dummies["popularity"].values

# Instantiate a ridge model
ridge = Ridge(alpha=0.2)

# Perform cross-validation
scores = cross_val_score(ridge, X, y, cv=kf, scoring="neg_mean_squared_error")

# Calculate RMSE
rmse = np.sqrt(-scores)
print("Average RMSE: {}".format(np.mean(rmse)))
```

```
## Average RMSE: 10.356167918309263
```

```
print("Standard Deviation of the target array: {}".format(np.std(y)))
```

```
## Standard Deviation of the target array: 14.02156909907019
```

Great work! An average RMSE of approximately `8.24` is lower than the standard deviation of the target variable (song popularity), suggesting the model is reasonably accurate.

Handling missing data

Dropping missing data

Over the next three exercises, you are going to tidy the `music_df` dataset. You will create a pipeline to impute missing values and build a KNN classifier model, then use it to predict whether a song is of the "`Rock`" genre.

In this exercise specifically, you will drop missing values accounting for less than 5% of the dataset, and convert the "`genre`" column into a binary feature.

- Print the number of missing values for each column in the `music_df` dataset, sorted in ascending order.
- Remove values for all columns with 50 or fewer missing values.
- Convert `music_df\["genre"\]` to values of `1` if the row contains "`Rock`", otherwise change the value to `0`.

```
# Print missing values for each column
print(music_df.isna().sum().sort_values())

# Remove values where less than 5% are missing
```

```
## popularity          0
## acousticness        0
## danceability        0
## duration_ms         0
## energy              0
## instrumentalness    0
## liveness            0
## loudness            0
## speechiness         0
## tempo               0
## valence             0
## genre               0
## dtype: int64
```

```
music_df = music_df.dropna(subset=["genre", "popularity", "loudness", "liveness",
"tempo"])

# Convert genre to a binary feature
music_df["genre"] = np.where(music_df["genre"] == "Rock", 1, 0)

print(music_df.isna().sum().sort_values())
```

```
## popularity          0
## acousticness        0
## danceability        0
## duration_ms         0
## energy              0
## instrumentalness    0
## liveness            0
## loudness            0
## speechiness         0
## tempo               0
## valence             0
## genre               0
## dtype: int64
```

```
print("Shape of the `music_df`: {}".format(music_df.shape))
```

```
## Shape of the `music_df`: (1000, 12)
```

Well done! The dataset has gone from 1000 observations down to 892, but it is now in the correct format for binary classification and the remaining missing values can be imputed as part of a pipeline.

Pipeline for song genre prediction: I

Now it's time to build a pipeline. It will contain steps to impute missing values using the mean for each feature and build a KNN model for the classification of song genre.

The modified `music_df` dataset that you created in the previous exercise has been preloaded for you, along with `KNeighborsClassifier` and `train_test_split`.

- Import `SimpleImputer` and `Pipeline`.
- Instantiate an imputer.
- Instantiate a KNN classifier with three neighbors.
- Create `steps`, a list of tuples containing the imputer variable you created, called “`imputer`”, followed by the `knn` model you created, called “`knn`”.

```
# Import modules
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline

# Instantiate an imputer
imputer = SimpleImputer()

# Instantiate a knn model
knn = KNeighborsClassifier(n_neighbors=3)

# Build steps for the pipeline
steps = [("imputer", imputer),
         ("knn", knn)]
```

Perfect pipeline skills! You are now ready to build and evaluate a song genre classification model.

Pipeline for song genre prediction: II

Having set up the steps of the pipeline in the previous exercise, you will now use it on the `music_df` dataset to classify the genre of songs. What makes pipelines so incredibly useful is the simple interface that they provide.

`X_train`, `X_test`, `y_train`, and `y_test` have been preloaded for you, and `confusion_matrix` has been imported from `sklearn.metrics`.

- Create a pipeline using the steps you previously defined.
- Fit the pipeline to the training data.
- Make predictions on the test set.
- Calculate and print the confusion matrix.

```
# edited/added
imp_mean = imputer
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=21)

steps = [("imputer", imp_mean),
        ("knn", knn)]

# Create the pipeline
pipeline = Pipeline(steps)

# Fit the pipeline to the training data
pipeline.fit(X_train, y_train)

# Make predictions on the test set
```

```
## Pipeline(steps=[('imputer', SimpleImputer()),
##                 ('knn', KNeighborsClassifier(n_neighbors=3))])
```

```
y_pred = pipeline.predict(X_test)

# Print the confusion matrix
print(confusion_matrix(y_test, y_pred))
```

```
## [[0 0 0 ... 0 0 0]
##  [0 0 0 ... 0 0 0]
##  [0 0 0 ... 0 0 0]
##  ...
##  [0 0 0 ... 0 0 0]
##  [0 0 0 ... 0 0 0]
##  [0 0 0 ... 0 0 0]]
```

Excellent! See how easy it is to scale our model building workflow using pipelines. In this case, the confusion matrix highlights that the model had 79 true positives and 82 true negatives!

Centering and scaling

Centering and scaling for regression

Now you have seen the benefits of scaling your data, you will use a pipeline to preprocess the `music_df` features and build a lasso regression model to predict a song's loudness.

`X_train`, `X_test`, `y_train`, and `y_test` have been created from the `music_df` dataset, where the target is `"loudness"` and the features are all other columns in the dataset. `Lasso` and `Pipeline` have also been imported for you.

Note that `"genre"` has been converted to a binary feature where `1` indicates a rock song, and `0` represents other genres.

- Import `StandardScaler`.
- Create the steps for the pipeline object, a `StandardScaler` object called `“scaler”`, and a lasso model called `“lasso”` with `alpha` set to `0.5`.
- Instantiate a pipeline with steps to scale and build a lasso regression model.
- Calculate the R-squared value on the test data.

```
# Import StandardScaler
from sklearn.preprocessing import StandardScaler

# Create pipeline steps
steps = [("scaler", StandardScaler()),
         ("lasso", Lasso(alpha=0.5))]

# Instantiate the pipeline
pipeline = Pipeline(steps)
pipeline.fit(X_train, y_train)

# Calculate and print R-squared
```

```
## Pipeline(steps=[('scaler', StandardScaler()), ('lasso', Lasso(alpha=0.5))])
```

```
print(pipeline.score(X_test, y_test))
```

```
## 0.47454082360792205
```

Awesome scaling! The model may have only produced an R-squared of `0.619`, but without scaling this exact model would have only produced a score of `0.35`, which proves just how powerful scaling can be!

Centering and scaling for classification

Now you will bring together scaling and model building into a pipeline for cross-validation.

Your task is to build a pipeline to scale features in the `music_df` dataset and perform grid search cross-validation using a logistic regression model with different values for the hyperparameter `C`. The target variable here is `“genre”`, which contains binary values for rock as `1` and any other genre as `0`.

`StandardScaler`, `LogisticRegression`, and `GridSearchCV` have all been imported for you.

- Build the steps for the pipeline: a `StandardScaler()` object named `“scaler”`, and a logistic regression model named `“logreg”`.
- Create the `parameters`, searching 20 equally spaced float values ranging from `0.001` to `1.0` for the logistic regression model’s `C` hyperparameter within the pipeline.
- Instantiate the grid search object.
- Fit the grid search object to the training data.

```
# Build the steps
steps = [("scaler", StandardScaler()),
         ("logreg", LogisticRegression())]
pipeline = Pipeline(steps)

# Create the parameter space
parameters = {"logreg__C": np.linspace(0.001, 1.0, 20)}
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=21)

# Instantiate the grid search object
cv = GridSearchCV(pipeline, param_grid=parameters)

# Fit to the training data
cv.fit(X_train, y_train)
```

```
## GridSearchCV(estimator=Pipeline(steps=[('scaler', StandardScaler()),
##                                        ('logreg', LogisticRegression())]),
##              param_grid={'logreg__C': array([0.001     , 0.05357895, 0.10615789,
0.15873684, 0.21131579,
##        0.26389474, 0.31647368, 0.36905263, 0.42163158, 0.47421053,
##        0.52678947, 0.57936842, 0.63194737, 0.68452632, 0.73710526,
##        0.78968421, 0.84226316, 0.89484211, 0.94742105, 1.        ])})
##
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/model_selection/_split.py:670: UserWarning: The least populated
class in y has only 1 members, which is less than n_splits=5.
##   warnings.warn(("The least populated class in y has only %d"
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to
converge (status=1):
## STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
##
## Increase the number of iterations (max_iter) or scale the data as shown in:
##     https://scikit-learn.org/stable/modules/preprocessing.html
## Please also refer to the documentation for alternative solver options:
##     https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
##   n_iter_i = _check_optimize_result(
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to
converge (status=1):
## STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
##
## Increase the number of iterations (max_iter) or scale the data as shown in:
##     https://scikit-learn.org/stable/modules/preprocessing.html
## Please also refer to the documentation for alternative solver options:
##     https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
##   n_iter_i = _check_optimize_result(
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to
converge (status=1):
## STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
##
## Increase the number of iterations (max_iter) or scale the data as shown in:
##     https://scikit-learn.org/stable/modules/preprocessing.html
## Please also refer to the documentation for alternative solver options:
##     https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
##   n_iter_i = _check_optimize_result(
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to
converge (status=1):
## STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
##
## Increase the number of iterations (max_iter) or scale the data as shown in:
##     https://scikit-learn.org/stable/modules/preprocessing.html
## Please also refer to the documentation for alternative solver options:
##     https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
##   n_iter_i = _check_optimize_result(
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to
converge (status=1):
## STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
##
## Increase the number of iterations (max_iter) or scale the data as shown in:
##     https://scikit-learn.org/stable/modules/preprocessing.html
## Please also refer to the documentation for alternative solver options:
##     https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
##   n_iter_i = _check_optimize_result(
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to
converge (status=1):
## STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
##
## Increase the number of iterations (max_iter) or scale the data as shown in:
##     https://scikit-learn.org/stable/modules/preprocessing.html
## Please also refer to the documentation for alternative solver options:
##     https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
##   n_iter_i = _check_optimize_result(
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to
converge (status=1):
## STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
##
## Increase the number of iterations (max_iter) or scale the data as shown in:
##     https://scikit-learn.org/stable/modules/preprocessing.html
## Please also refer to the documentation for alternative solver options:
##     https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
##   n_iter_i = _check_optimize_result(
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to
converge (status=1):
```

```
## STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
##
## Increase the number of iterations (max_iter) or scale the data as shown in:
##     https://scikit-learn.org/stable/modules/preprocessing.html
## Please also refer to the documentation for alternative solver options:
##     https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
##   n_iter_i = _check_optimize_result(
```

```
print(cv.best_score_, "\n", cv.best_params_)
```

```
## 0.0525000000000000005
##  {'logreg__C': 0.1061578947368421}
```

Well done! Using a pipeline shows that a logistic regression model with `“C”` set to approximately `0.1` produces a model with `0.8425` accuracy!

Evaluating multiple models

Visualizing regression model performance

Now you have seen how to evaluate multiple models out of the box, you will build three regression models to predict a song’s `“energy”` levels.

The `music_df` dataset has had dummy variables for `“genre”` added. Also, feature and target arrays have been created, and these have been split into `X_train`, `X_test`, `y_train`, and `y_test`.

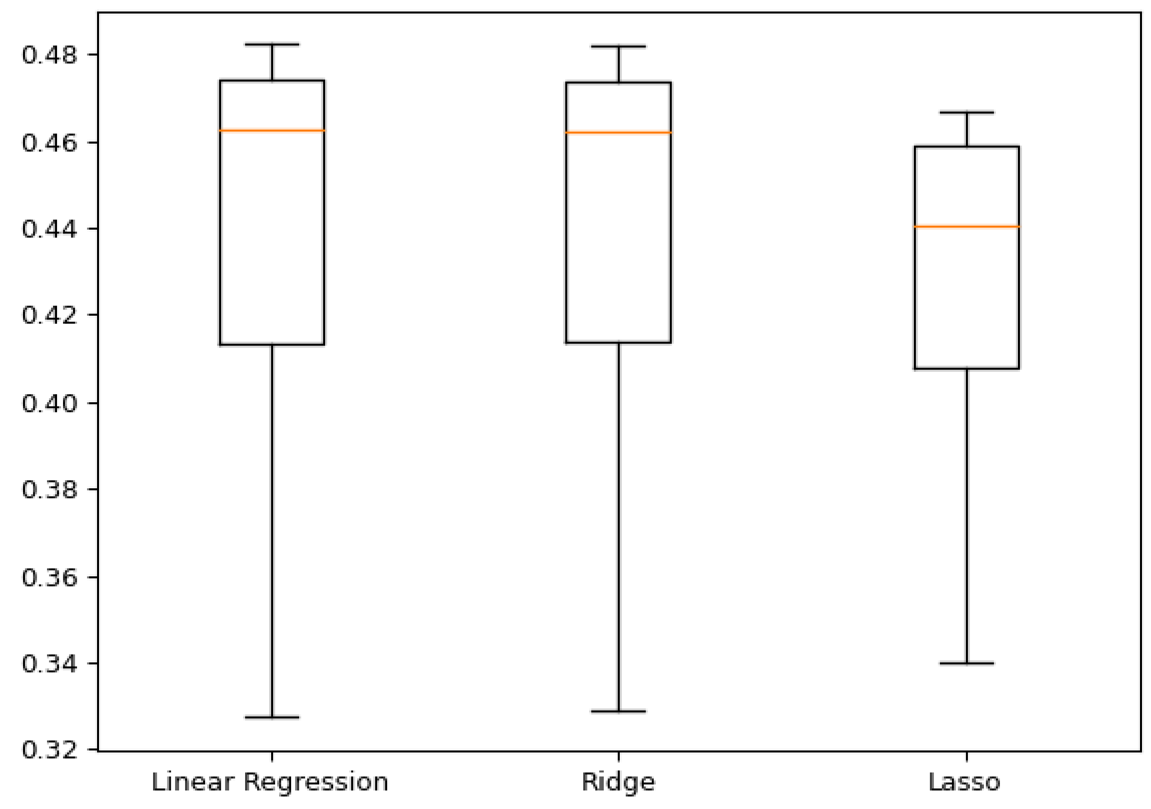
The following have been imported for you: `LinearRegression`, `Ridge`, `Lasso`, `cross_val_score`, and `KFold`.

- Write a for loop using `model` as the iterator, and `model.values()` as the iterable.
- Perform cross-validation on the training features and the training target array using the model, setting `cv` equal to the `KFold` object.
- Append the model’s cross-validation scores to the results list.
- Create a box plot displaying the results, with the x-axis labels as the names of the models.

```
models = {"Linear Regression": LinearRegression(), "Ridge": Ridge(alpha=0.1),
"Lasso": Lasso(alpha=0.1)}
results = []

# Loop through the models' values
for model in models.values():
  kf = KFold(n_splits=6, random_state=42, shuffle=True)

  # Perform cross-validation
  cv_scores = cross_val_score(model, X_train, y_train, cv=kf)

  # Append the results
  results.append(cv_scores)

# Create a box plot of the results
plt.boxplot(results, labels=models.keys())
```

```
## {'whiskers': [<matplotlib.lines.Line2D object at 0x7ffca8ceaa60>,
<matplotlib.lines.Line2D object at 0x7ffca8ceadf0>, <matplotlib.lines.Line2D object
at 0x7ffca8d2a2b0>, <matplotlib.lines.Line2D object at 0x7ffca8d2a640>,
<matplotlib.lines.Line2D object at 0x7ffca8d35be0>, <matplotlib.lines.Line2D object
at 0x7ffca8d35f70>], 'caps': [<matplotlib.lines.Line2D object at 0x7ffca8cff0a0>,
<matplotlib.lines.Line2D object at 0x7ffca8cff430>, <matplotlib.lines.Line2D object
at 0x7ffca8d2a9d0>, <matplotlib.lines.Line2D object at 0x7ffca8d2ad60>,
<matplotlib.lines.Line2D object at 0x7ffca8d3f340>, <matplotlib.lines.Line2D object
at 0x7ffca8d3f6d0>], 'boxes': [<matplotlib.lines.Line2D object at 0x7ffca8cd9c40>,
<matplotlib.lines.Line2D object at 0x7ffca8cffee0>, <matplotlib.lines.Line2D object
at 0x7ffca8d35850>], 'medians': [<matplotlib.lines.Line2D object at 0x7ffca8cff7c0>,
<matplotlib.lines.Line2D object at 0x7ffca8d35130>, <matplotlib.lines.Line2D object
at 0x7ffca8d3fa60>], 'fliers': [<matplotlib.lines.Line2D object at 0x7ffca8cffb50>,
<matplotlib.lines.Line2D object at 0x7ffca8d354c0>, <matplotlib.lines.Line2D object
at 0x7ffca8d3fdf0>], 'means': []}
```

```
plt.show()
```



Nicely done! Lasso regression is not a good model for this problem, while linear regression and ridge perform fairly equally. Let's make predictions on the test set, and see if the RMSE can guide us on model selection.

Predicting on the test set

In the last exercise, linear regression and ridge appeared to produce similar results. It would be appropriate to select either of those models; however, you can check predictive performance on the test set to see if either one can outperform the other.

You will use root mean squared error (RMSE) as the metric. The dictionary `models`, containing the names and instances of the two models, has been preloaded for you along with the training and target arrays `X_train_scaled`, `X_test_scaled`, `y_train`, and `y_test`.

- Import `mean_squared_error`.
- Fit the model to the scaled training features and the training labels.
- Make predictions using the scaled test features.
- Calculate RMSE by passing the test set labels and the predicted labels.

```
# edited/added
from sklearn.preprocessing import scale
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=21)
X_train_scaled = scale(X_train)
X_test_scaled = scale(X_test)

# Import mean_squared_error
from sklearn.metrics import mean_squared_error

for name, model in models.items():

  # Fit the model to the training data
  model.fit(X_train_scaled, y_train)

  # Make predictions on the test set
  y_pred = model.predict(X_test_scaled)

  # Calculate the test_rmse
  test_rmse = mean_squared_error(y_test, y_pred, squared=False)
  print("{} Test Set RMSE: {}".format(name, test_rmse))
```

```
## LinearRegression()
## Linear Regression Test Set RMSE: 10.368037372390296
## Ridge(alpha=0.1)
## Ridge Test Set RMSE: 10.367981364747633
## Lasso(alpha=0.1)
## Lasso Test Set RMSE: 10.354423827065155
```

The linear regression model just edges the best performance, although the difference is a RMSE of 0.00001 for popularity! Now let’s look at classification model selection.

Visualizing classification model performance

In this exercise, you will be solving a classification problem where the “popularity” column in the music_df dataset has been converted to binary values, with 1 representing popularity more than or equal to the median for the “popularity” column, and 0 indicating popularity below the median.

Your task is to build and visualize the results of three different models to classify whether a song is popular or not.

The data has been split, scaled, and preloaded for you as X_train_scaled, X_test_scaled, y_train, and y_test. Additionally, KNeighborsClassifier, DecisionTreeClassifier, and LogisticRegression have been imported.

- Create a dictionary of “Logistic Regression”, “KNN”, and “Decision Tree Classifier”, setting the dictionary’s values to a call of each model.
- Loop through the values in models.
- Instantiate a KFold object to perform 6 splits, setting shuffle to True and random_state to 12.
- Perform cross-validation using the model, the scaled training features, the target training set, and setting cv equal to kf.

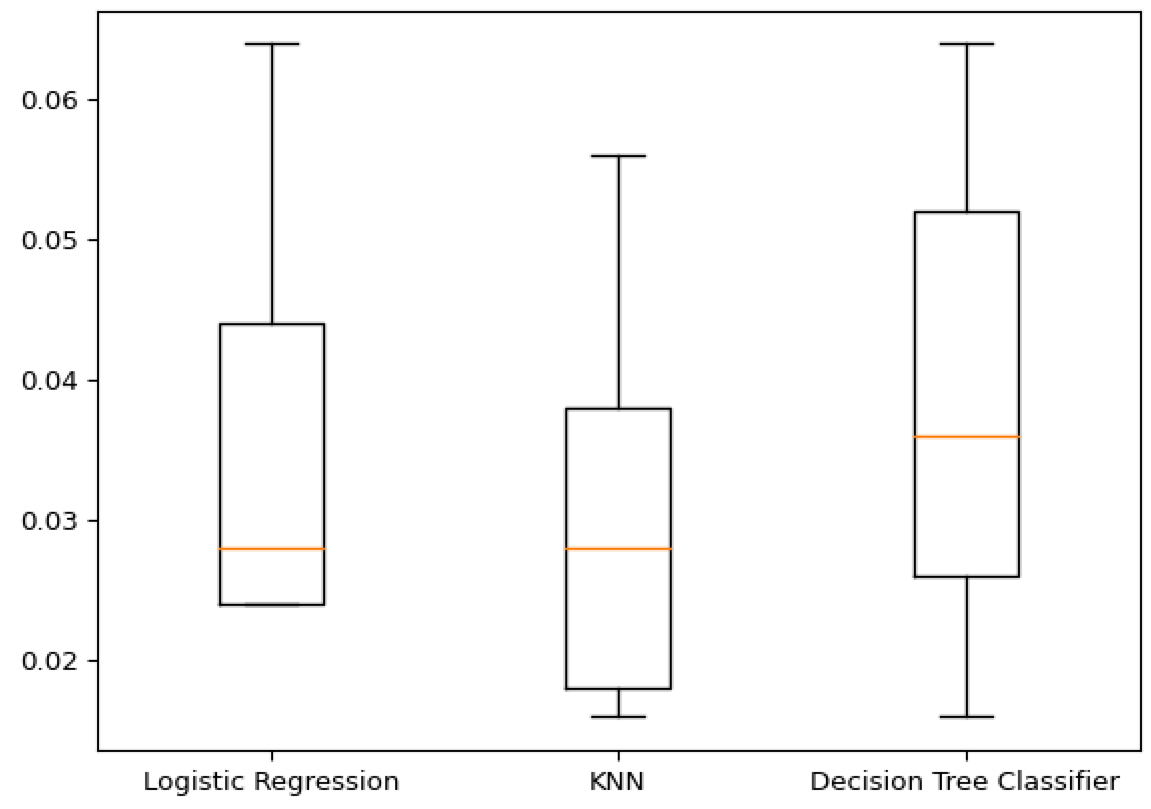
```
# edited/added
from sklearn.tree import DecisionTreeClassifier

# Create models dictionary
models = {"Logistic Regression": LogisticRegression(), "KNN": KNeighborsClassifier(),
"Decision Tree Classifier": DecisionTreeClassifier()}
results = []

# Loop through the models' values
for model in models.values():

  # Instantiate a KFold object
  kf = KFold(n_splits=6, random_state=12, shuffle=True)

  # Perform cross-validation
  cv_results = cross_val_score(model, X_train_scaled, y_train, cv=kf)
  results.append(cv_results)
```

```
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to
converge (status=1):
## STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
##
## Increase the number of iterations (max_iter) or scale the data as shown in:
##     https://scikit-learn.org/stable/modules/preprocessing.html
## Please also refer to the documentation for alternative solver options:
##     https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
##   n_iter_i = _check_optimize_result(
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to
converge (status=1):
## STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
##
## Increase the number of iterations (max_iter) or scale the data as shown in:
##     https://scikit-learn.org/stable/modules/preprocessing.html
## Please also refer to the documentation for alternative solver options:
##     https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
##   n_iter_i = _check_optimize_result(
```

```
plt.boxplot(results, labels=models.keys())
```

```
## {'whiskers': [<matplotlib.lines.Line2D object at 0x7ffca68f9d30>,
<matplotlib.lines.Line2D object at 0x7ffca68f9fa0>, <matplotlib.lines.Line2D object
at 0x7ffca6910580>, <matplotlib.lines.Line2D object at 0x7ffca6910910>,
<matplotlib.lines.Line2D object at 0x7ffca2d8beb0>, <matplotlib.lines.Line2D object
at 0x7ffca2d94280>], 'caps': [<matplotlib.lines.Line2D object at 0x7ffca6907370>,
<matplotlib.lines.Line2D object at 0x7ffca6907700>, <matplotlib.lines.Line2D object
at 0x7ffca6910ca0>, <matplotlib.lines.Line2D object at 0x7ffca2d8b070>,
<matplotlib.lines.Line2D object at 0x7ffca2d94610>, <matplotlib.lines.Line2D object
at 0x7ffca2d949a0>], 'boxes': [<matplotlib.lines.Line2D object at 0x7ffca68f99a0>,
<matplotlib.lines.Line2D object at 0x7ffca69101f0>, <matplotlib.lines.Line2D object
at 0x7ffca2d8bb20>], 'medians': [<matplotlib.lines.Line2D object at 0x7ffca6907a90>,
<matplotlib.lines.Line2D object at 0x7ffca2d8b400>, <matplotlib.lines.Line2D object
at 0x7ffca2d94d30>], 'fliers': [<matplotlib.lines.Line2D object at 0x7ffca6907e20>,
<matplotlib.lines.Line2D object at 0x7ffca2d8b790>, <matplotlib.lines.Line2D object
at 0x7ffca2da1100>], 'means': []}
```

```
plt.show()
```



Looks like logistic regression is the best candidate based on the cross-validation results! Let's wrap up by building a pipeline

Pipeline for predicting song popularity

For the final exercise, you will build a pipeline to impute missing values, scale features, and perform hyperparameter tuning of a logistic regression model. The aim is to find the best parameters and accuracy when predicting song genre!

All the models and objects required to build the pipeline have been preloaded for you.

- Create the steps for the pipeline by calling a simple imputer, a standard scaler, and a logistic regression model.
- Create a pipeline object, and pass the `steps` variable.
- Instantiate a grid search object to perform cross-validation using the pipeline and the parameters.
- Print the best parameters and compute and print the test set accuracy score for the grid search object.

```python
# Create steps
steps = [("imp_mean", SimpleImputer()),
         ("scaler", StandardScaler()),
         ("logreg", LogisticRegression())]

# Set up pipeline
pipeline = Pipeline(steps)
params = {"logreg__solver": ["newton-cg", "saga", "lbfgs"],
         "logreg__C": np.linspace(0.001, 1.0, 10)}

# Create the GridSearchCV object
tuning = GridSearchCV(pipeline, param_grid=params)
tuning.fit(X_train, y_train)
```

```
## GridSearchCV(estimator=Pipeline(steps=[('imp_mean', SimpleImputer()),
##                                        ('scaler', StandardScaler()),
##                                        ('logreg', LogisticRegression())]),
##              param_grid={'logreg__C': array([0.001, 0.112, 0.223, 0.334, 0.445,
0.556, 0.667, 0.778, 0.889,
##        1.   ]),
##                          'logreg__solver': ['newton-cg', 'saga', 'lbfgs']})
##
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/model_selection/_split.py:670: UserWarning: The least populated
class in y has only 1 members, which is less than n_splits=5.
##   warnings.warn(("The least populated class in y has only %d"
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to
converge (status=1):
## STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
##
## Increase the number of iterations (max_iter) or scale the data as shown in:
##     https://scikit-learn.org/stable/modules/preprocessing.html
## Please also refer to the documentation for alternative solver options:
##     https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
##   n_iter_i = _check_optimize_result(
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
```

```
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to
converge (status=1):
## STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
##
```

```
## Increase the number of iterations (max_iter) or scale the data as shown in:
##     https://scikit-learn.org/stable/modules/preprocessing.html
## Please also refer to the documentation for alternative solver options:
##     https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
##   n_iter_i = _check_optimize_result(
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_sag.py:329: ConvergenceWarning: The max_iter was
reached which means the coef_ did not converge
##   warnings.warn("The max_iter was reached which means "
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to
converge (status=1):
## STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
##
## Increase the number of iterations (max_iter) or scale the data as shown in:
##     https://scikit-learn.org/stable/modules/preprocessing.html
## Please also refer to the documentation for alternative solver options:
##     https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
##   n_iter_i = _check_optimize_result(
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to
converge (status=1):
## STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
##
## Increase the number of iterations (max_iter) or scale the data as shown in:
##     https://scikit-learn.org/stable/modules/preprocessing.html
## Please also refer to the documentation for alternative solver options:
##     https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
##   n_iter_i = _check_optimize_result(
## /Users/macos/Library/r-miniconda/envs/r-reticulate/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:762: ConvergenceWarning: lbfgs failed to
converge (status=1):
## STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
##
## Increase the number of iterations (max_iter) or scale the data as shown in:
##     https://scikit-learn.org/stable/modules/preprocessing.html
## Please also refer to the documentation for alternative solver options:
##     https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
##   n_iter_i = _check_optimize_result(
```

```python
y_pred = tuning.predict(X_test)

# Compute and print performance
print("Tuned Logistic Regression Parameters: {}, Accuracy:
{}".format(tuning.best_params_, tuning.score(X_test, y_test)))
```

```
## Tuned Logistic Regression Parameters: {'logreg__C': 0.112, 'logreg__solver':
'newton-cg'}, Accuracy: 0.056
```

Excellent - you've selected a model, built a preprocessing pipeline, and performed hyperparameter tuning to create a model that is 82% accurate in predicting song genres!

Congratulations

Congratulations

Well done on completing the course, I predicted that you would!

What you've covered

To recap, you have learned the fundamentals of using supervised learning techniques to build predictive models for both regression and classification problems. You have learned the concepts of underfitting and overfitting, how to split data, and perform cross-validation.

What you've covered

You also learned about data preprocessing techniques, selected which model to build, performed hyperparameter tuning, assessed model performance, and used pipelines!

Where to go from here?

We covered several models, but there are plenty of others, so to learn more we recommend checking out some of our courses. We also have courses that dive deeper into topics we introduced, such as preprocessing, or model validation. There are other courses on topics we did not cover, such as feature engineering, and unsupervised learning. Additionally, we have many machine learning projects where you can apply the skills you've learned here!

Thank you!

Congratulations again, and thank you for taking the course! I hope you enjoy using scikit-learn for your supervised learning problems from now on!