HEXAD01

SOFTWARE DEVELOPMENT TEAM

Aidan Zorbas
Dawson Brown
Jenny Yu
Jingrun Long
Kara Autumn Jiang
Vanessa Pierre

Scikit-learn Easy Bugs Report

Assignment 2 Deliverable

Table of Contents

Scikit-learn Issues

Verifying Issues

Listed below are five issues in the scikit-learn codebase that our team has identified. In order to demonstrate the behaviour of these issues, we have created an interactive program in our repository named **sklearn_bugs_demo.py.**

To verify each of these issues, use Anaconda to install scikit-learn (Please do not build from our repository as source since this codebase contains fixes for two of these components).

Once scikit-learn has been installed, run the **sklearn_bugs_demo** program and select a component to test. For each component, the program will display the input and expected output, as well as the actual output. You can select the Q option at any time to exit the program.

List of Issues Identified

1. [Issue 22478] DummyRegressor

Description of Issue

Some parameters passed into the DummyRegressor are erroneously converted into Numpy format after the fit method is invoked.

Related Components

sklearn/Dummy.py -> DummyRegressor

Source

DummyRegressor converts some params to NumPy after fit() #22478

2. [Issue 19352] IterativeImputer

Description of Issue

When setting the estimator as PLSRegression(), a ValueError is triggered by the module '_interactive.py' in line 348 because of a shape mismatch.

Potentially Related Components

- sklearn/cross_decomposition/_pls.py/PLSRegression
- PLSRegression: partial least squares regression
- sklearn/impute/_iterative.py ->IterativeImputer

- Multivariate imputer that estimates each feature from all the others: A strategy for imputing missing values by modelling each feature with missing values as a function of other features in a round-robin fashion. (sklearn docs)
- Imputation is a technique used for replacing the missing data with some substitute value to retain most of the data/information of the dataset. (src)

Source

Interactive Imputer cannot accept PLSRegression() as an estimator due to "shape mismatch" #19352

3. [Issue 21207] CountVectorizer

Description of Issue

Some characters are transformed into uppercase despite the lowercase attribute being set to True. This then yields warning messages: “UserWarning: Upper case characters found in vocabulary while ‘lowercase’ is True.”

Related Components

sklearn/feature_extraction/text.py -> CountVectorizer

Source

CountVectorizer(lowercase=True,strip_accents='unicode') may produce vocab that contains uppercase chars #21207

4. [Issue 19693] Ridge and Lasso

Description of Issue

Ridge.coef_ returns an array with shape (1, n_features), while Lasso.coef_ returns an array with shape (n_features,).

Potentially Related Components

sklearn/linear_model/_Ridge.py -> Ridge
sklearn/linear_model/_coordinate_descent.py -> Lasso

Source

Ridge and Lasso return different shaped .coef_ attributes #19693

5. [Issue 18941] fit.transform and fit_transform Inconsistency

Description of Issue

PCA’s fit_transform returns a different result than applying fit and transform individually.

Potentially Related Components

sklearn/decomposition/_pca.py -> PCA

Source

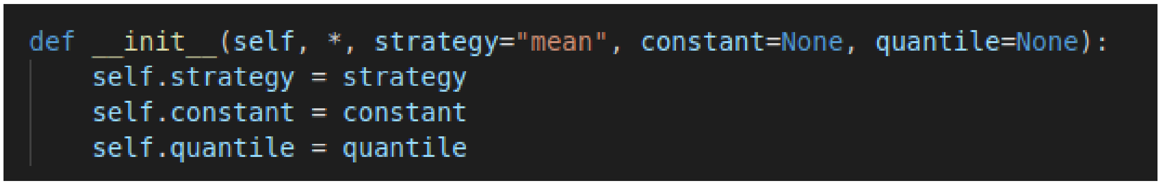
.fit.transform != .fit_transform inconsistency in PCA results #18941

Bugfix 1: DummyRegressor [Issue 22478]

Source: DummyRegressor converts some params to NumPy after fit() #22478

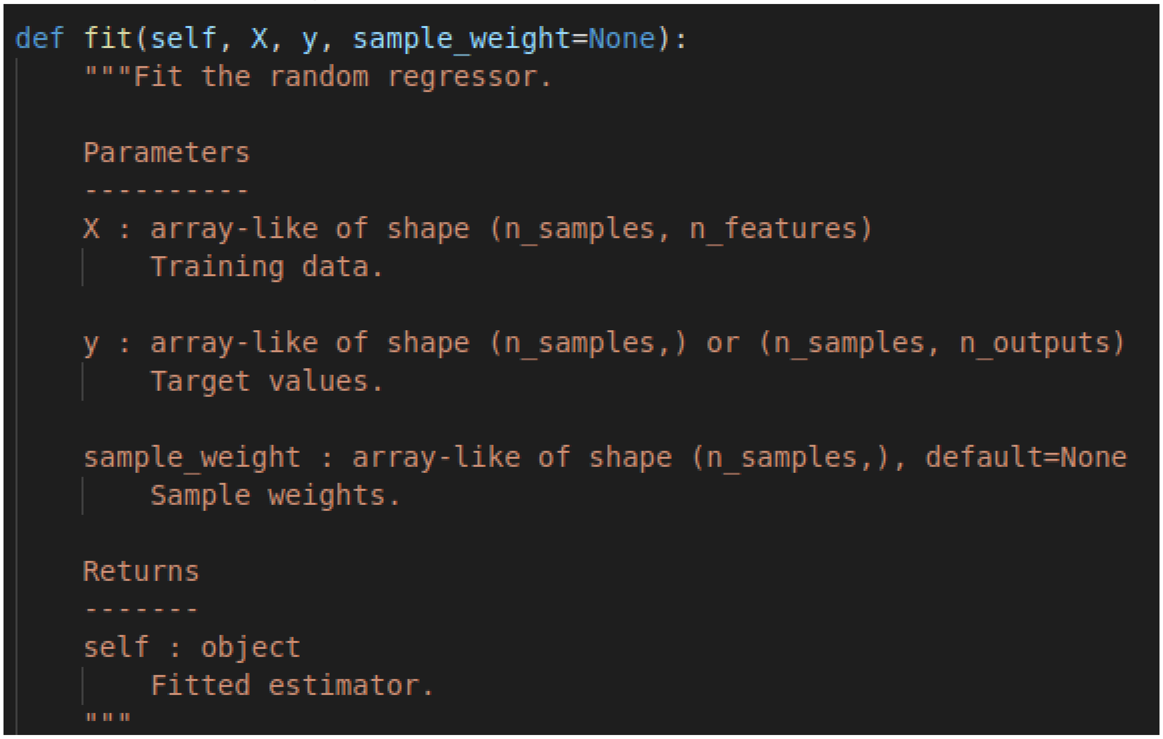
Explanation

Scikit-learn’s DummyRegressor class
(https://github.com/ajz2000/scikit-learn/blob/main/sklearn/dummy.py)
contains an attribute “constant”, which is supposed to hold an int or float or array-like of shape (n_outputs,)



```
def __init__(self, *, strategy="mean", constant=None, quantile=None):
    self.strategy = strategy
    self.constant = constant
    self.quantile = quantile
```

This value is used during calculations made in the “fit” method.

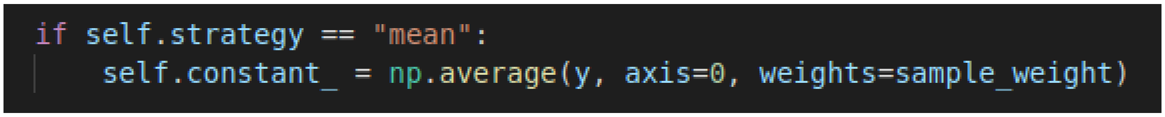


```
def fit(self, X, y, sample_weight=None):
    """Fit the random regressor.

    Parameters
    ----------
    X : array-like of shape (n_samples, n_features)
        Training data.

    y : array-like of shape (n_samples,) or (n_samples, n_outputs)
        Target values.

    sample_weight : array-like of shape (n_samples,), default=None
        Sample weights.

    Returns
    -------
    self : object
        Fitted estimator.
    """
```

After running “fit”, “constant” is converted into NumPy format, which violates scikit-learn API.

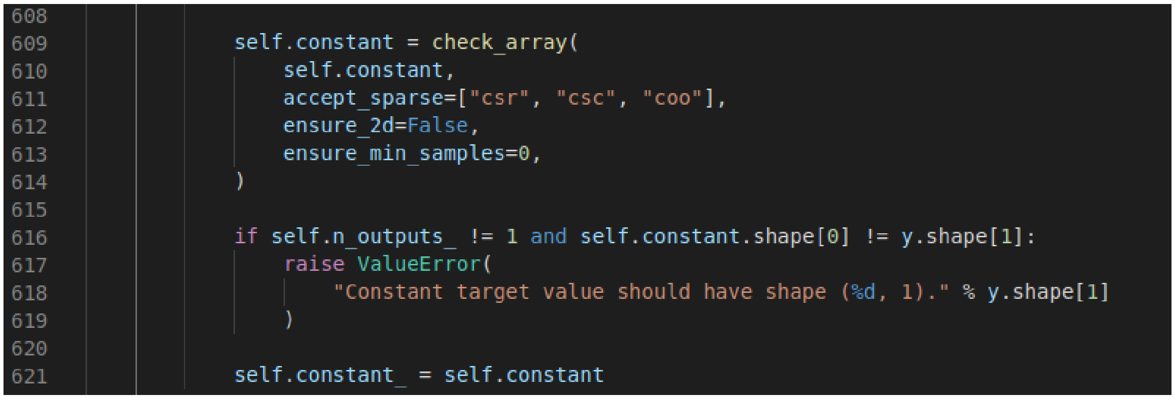
Our Fix

Inside “fit”, a similarly named attribute “constant_” is also used, which is an array typed variable



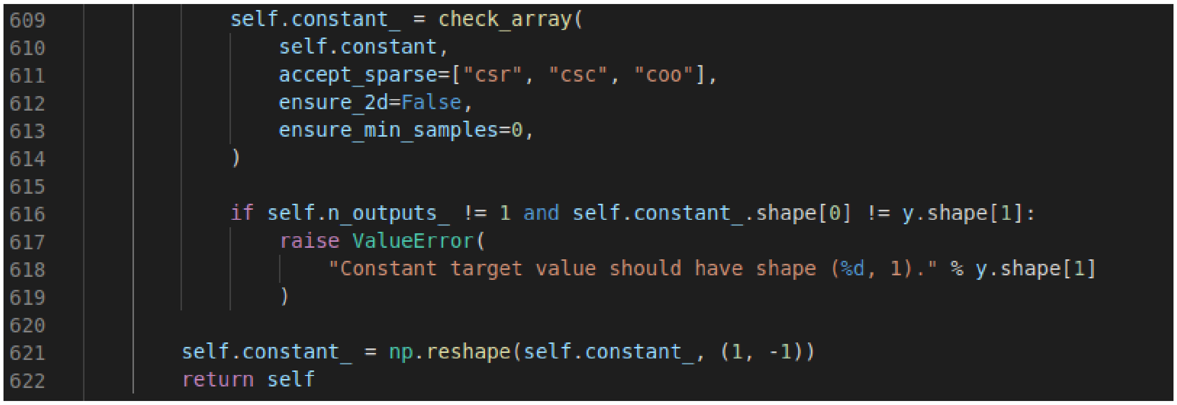
```
if self.strategy == "mean":
    self.constant_ = np.average(y, axis=0, weights=sample_weight)
```

When the constant fit strategy is used with the “fit” method, the normal “constant” attribute is set to the result of check_array, rather than “constant_” (line 609). The result is then used in line 616, and “constant_” is finally set to the value of “constant” on line 621.



```
608
609             self.constant = check_array(
610                 self.constant,
611                 accept_sparse=["csr", "csc", "coo"],
612                 ensure_2d=False,
613                 ensure_min_samples=0,
614             )
615
616             if self.n_outputs_ != 1 and self.constant.shape[0] != y.shape[1]:
617                 raise ValueError(
618                     "Constant target value should have shape (%d, 1)." % y.shape[1]
619                 )
620
621             self.constant_ = self.constant
```

By replacing “constant” with “constant_” on lines 609 and 616, and removing line 621, the type of “constant” remains unchanged (note that check_array is simply performing validation), and the function’s behaviour and final value of “constant_” remain the same. This fixes the bug.



```
609             self.constant_ = check_array(
610                 self.constant,
611                 accept_sparse=["csr", "csc", "coo"],
612                 ensure_2d=False,
613                 ensure_min_samples=0,
614             )
615
616             if self.n_outputs_ != 1 and self.constant_.shape[0] != y.shape[1]:
617                 raise ValueError(
618                     "Constant target value should have shape (%d, 1)." % y.shape[1]
619                 )
620
621         self.constant_ = np.reshape(self.constant_, (1, -1))
622         return self
```

Test Cases Summary

Test Number	**Description (strategy always set to constant)**
Test Set 1	Zero Float Constant
Test Set 2	Zero Integer Constant
Test Set 3	Positive Integer Constant
Test Set 4	Negative Integer Constant
Test Set 5	Negative Float Constant Part 1
Test Set 6	Negative Float Constant Part 2 (Uses different X and y arrays)
Test Set 7	Large Positive Float Constant (Uses different X and y arrays)
Test Set 8	Large Negative Float Constant (Uses different X and y arrays)

Test Set 9	Small Positive Float Constant (Uses different X and y arrays)
Test Set 10	Small Negative Float Constant (Uses different X and y arrays)

Test Coverage and Acceptance Conditions

The test suite ensures that the attribute “constant” has the correct typing and value before and after the fit method is invoked. The table above outlines the diverse set of ranges tested for the attribute “constant”. For each set, the estimator’s "strategy" attribute is set to "constant" and a unique value for the "constant" attribute is passed in. Test sets 6 - 10 also use a different data set to fit the estimator. When building scikit-learn from source with the bugfix applied, all test sets defined above pass, and additionally, the existing scikit-learn test suite also passes.

Running Our Test Cases

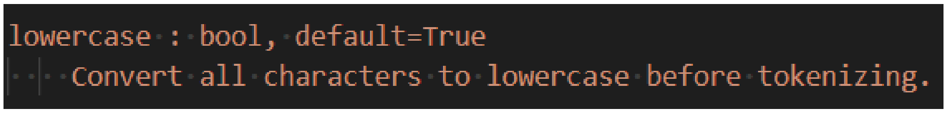
Our test cases have been included within a dedicated testing folder within our A2 directory, and the modified files have also been included for clarity. To run our test cases, please first build scikit-learn from source using the main branch of our fork (https://github.com/ajz2000/scikit-learn/), and then run the testDummyRegressor.py script. This should verify that our bugfix both passes the built-in scikit-learn unit tests, as well as the test suite that we have generated.

Bugfix 2: CountVectorizer [Issue #21207]

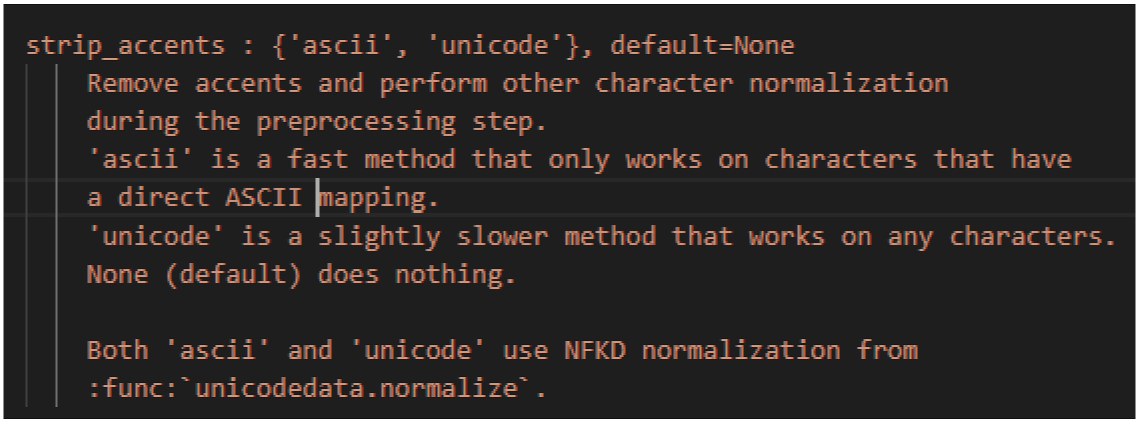
Source: CountVectorizer(lowercase=True,strip_accents='unicode') may produce vocab that contains uppercase chars #21207

Explanation

The VectorizerMixin subclasses (CountVectorizer and HashVectorizer) in sklearn/feature_extraction/text.py are used for processing text documents, but the lowercase parameter which “converts all characters to lowercase before tokenizing” does not always correctly work in the case of text with Unicode characters when the strip_accents function is set to ‘unicode’.



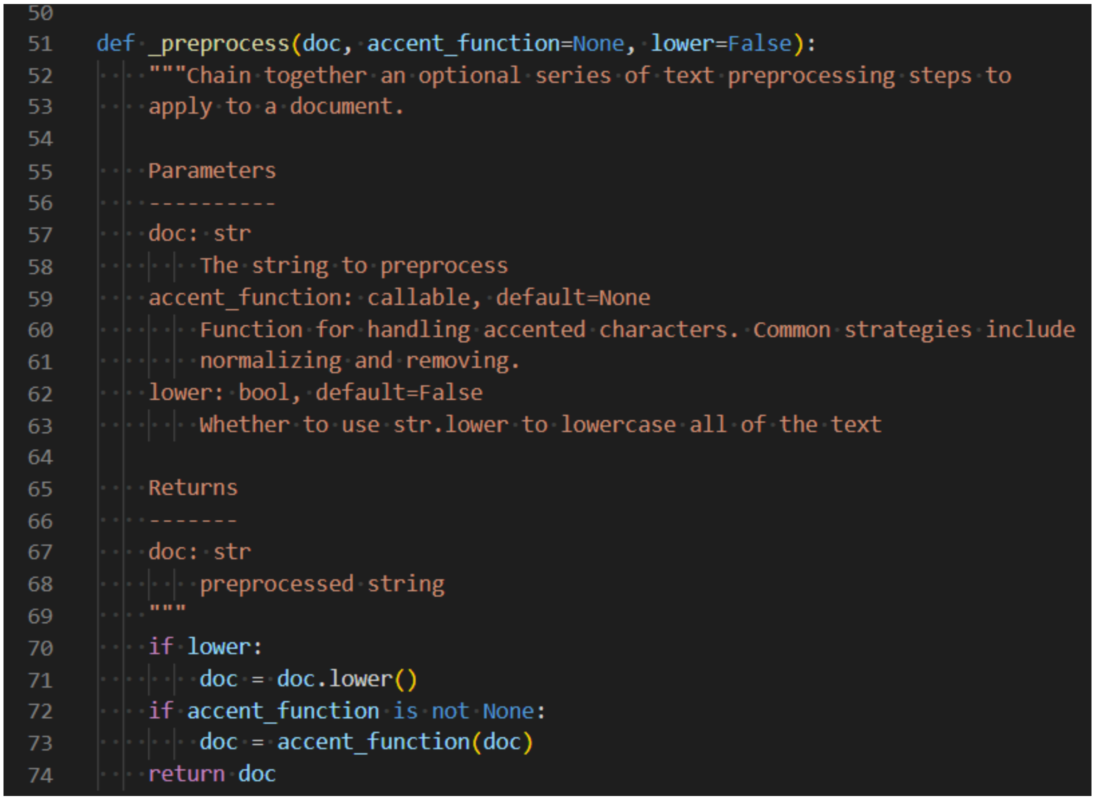
```
lowercase : bool, default=True
    Convert all characters to lowercase before tokenizing.
```



```
strip_accents : {'ascii', 'unicode'}, default=None
    Remove accents and perform other character normalization
    during the preprocessing step.
    'ascii' is a fast method that only works on characters that have
    a direct ASCII mapping.
    'unicode' is a slightly slower method that works on any characters.
    None (default) does nothing.

    Both 'ascii' and 'unicode' use NFKD normalization from
    :func:`unicodedata.normalize`.
```

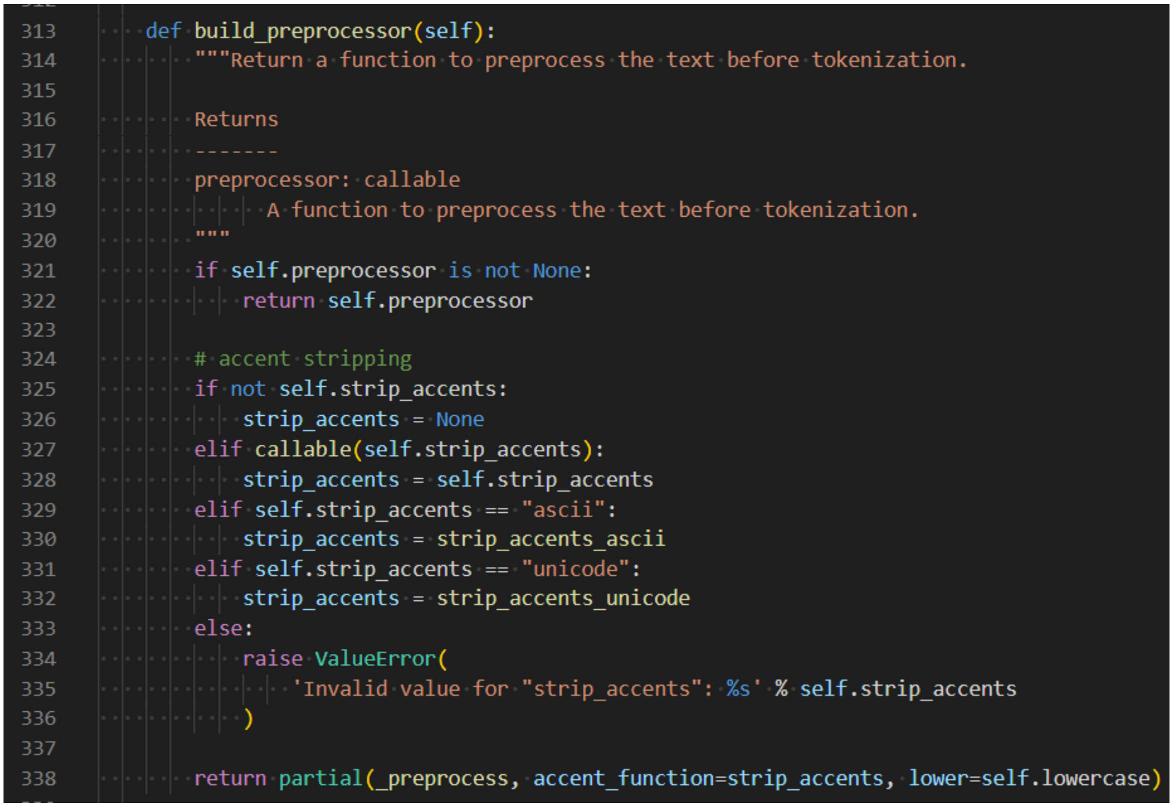
Our Fix

The reason for this bug is because in _preprocess() which both classes use (CountVectorizer -> _count_vocab -> build_analyzer -> build_preprocessor ->_preprocess, HashVectorizer -> transform -> build_analyzer -> build_preprocessor -> _preprocess), the input text document is set to lowercase before applying the chosen accent_function.



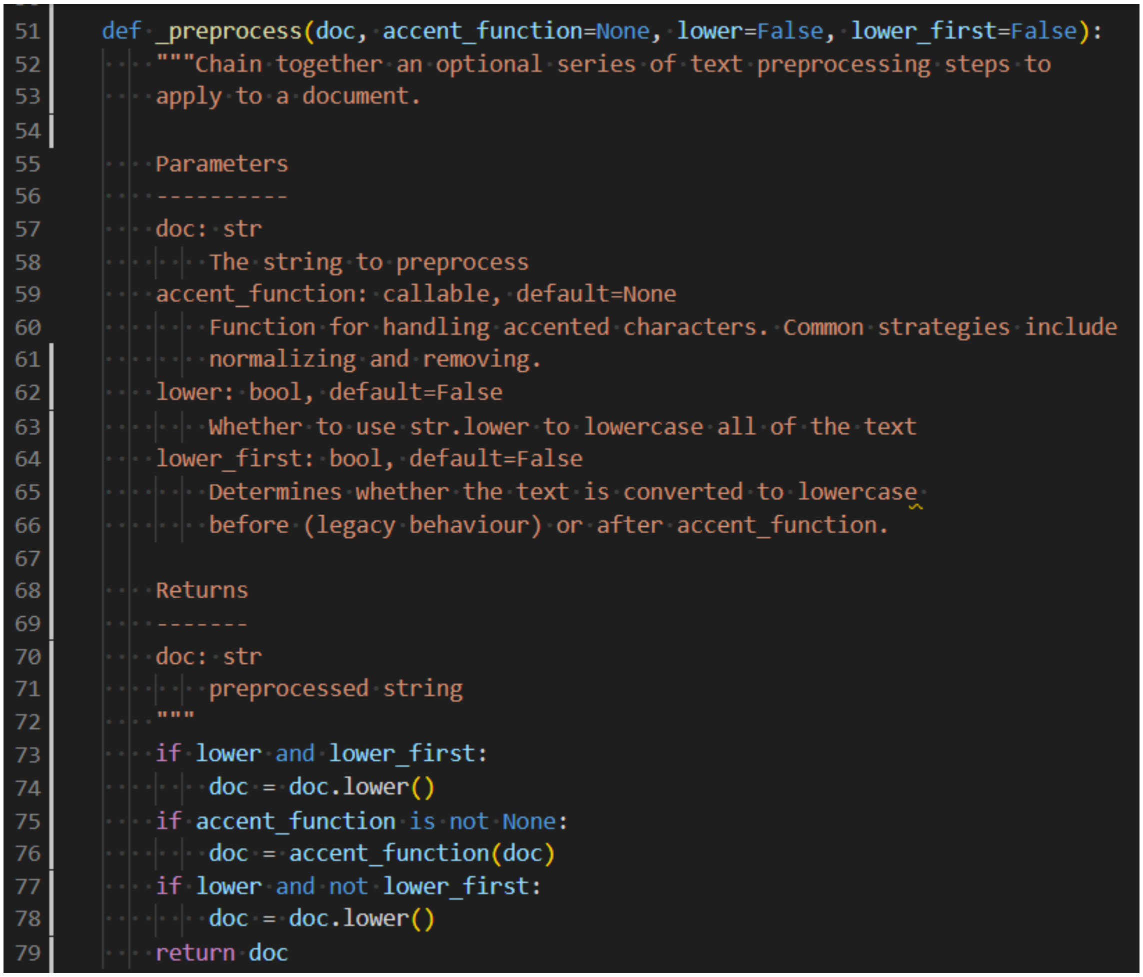
```
50
51  def _preprocess(doc, accent_function=None, lower=False):
52      """Chain together an optional series of text preprocessing steps to
53      apply to a document.
54
55      Parameters
56      ----------
57      doc: str
58          The string to preprocess
59      accent_function: callable, default=None
60          Function for handling accented characters. Common strategies include
61          normalizing and removing.
62      lower: bool, default=False
63          Whether to use str.lower to lowercase all of the text
64
65      Returns
66      -------
67      doc: str
68          preprocessed string
69      """
70      if lower:
71          doc = doc.lower()
72      if accent_function is not None:
73          doc = accent_function(doc)
74      return doc
```

As seen above, at line 71 the document is set to lowercase before the accent_function is applied.



```
    def build_preprocessor(self):
        """Return a function to preprocess the text before tokenization.

        Returns
        -------
        preprocessor: callable
              A function to preprocess the text before tokenization.
        """
        if self.preprocessor is not None:
            return self.preprocessor

        # accent stripping
        if not self.strip_accents:
            strip_accents = None
        elif callable(self.strip_accents):
            strip_accents = self.strip_accents
        elif self.strip_accents == "ascii":
            strip_accents = strip_accents_ascii
        elif self.strip_accents == "unicode":
            strip_accents = strip_accents_unicode
        else:
            raise ValueError(
                'Invalid value for "strip_accents": %s' % self.strip_accents
            )

        return partial(_preprocess, accent_function=strip_accents, lower=self.lowercase)
```

This means when the accent_function is set to strip_accents_unicode like at line 332 in build_preprocessor(), it may potentially result in characters that are uppercase instead of only having lowercase letters in the final result



```
def _preprocess(doc, accent_function=None, lower=False, lower_first=False):
    """Chain together an optional series of text preprocessing steps to
    apply to a document.

    Parameters
    ----------
    doc: str
        The string to preprocess
    accent_function: callable, default=None
        Function for handling accented characters. Common strategies include
        normalizing and removing.
    lower: bool, default=False
        Whether to use str.lower to lowercase all of the text
    lower_first: bool, default=False
        Determines whether the text is converted to lowercase
        before (legacy behaviour) or after accent_function.

    Returns
    -------
    doc: str
        preprocessed string
    """
    if lower and lower_first:
        doc = doc.lower()
    if accent_function is not None:
        doc = accent_function(doc)
    if lower and not lower_first:
        doc = doc.lower()
    return doc
```

To fix this, the conversion to lowercase needs to occur after the accents are transformed into their normalised forms. The only problem is that changing this behaviour may break existing apps which already account for the current behaviour of the app. To account for this, we decided to add a parameter to let the user decide when to set the input to lowercase. By default, the function uses the new behaviour, ensuring all characters are properly converted to lowercase. However, the user can override this by setting the lower_first parameter to true to restore the old behaviour.

Test Cases Summary

Test Number	Description
Test Set 1	lowercase=True, lower_first=False (New Behaviour)
Test Set 2	lowercase=False, lower_first=False (New Behaviour)
Test Set 3	lowercase=True, lower_first=True (Old Behaviour)

Test Set 4	lowercase=False, lower_first=True (Old Behaviour)
Test Set 5	No Unicode characters in input string
Test Set 6	Empty string
Test Set 7	1 non-accented ASCII character
Test Set 8	Mix of non-accented ASCII and normal characters, lowercase=True
Test Set 9	Only ASCII characters (accented and unaccented)
Test Set 10	Only Unicode characters (accented and unaccented)
Test Set 11	Mix of non-accented ASCII and normal characters, lowercase=False
Test Set 12	Mix of accented ASCII and normal characters

Test Coverage and Acceptance Conditions

For each test case: a CountVectorizer instance is created with the test case's specified parameters, build_analyzer() is called which calls build_preprocessor() and _preprocess(), and analyze() is called for each token of the test document. The results are then compared to the expected output.Tests 1-4 cover all combinations of lowercase and lower_first for the input from the original issue. Tests 5-12 cover various string test cases for ASCII and Unicode characters, and run under the new behaviour with lower_first=False. The bug fix results in each of the test cases passing, and does not break other test cases during scikit-learn build from source.

Running Our Test Cases

Our test cases have been included within a dedicated testing folder within our A2 directory, and the modified files have also been included for clarity. To run our test cases, please first build scikit-learn from source using the main branch of our fork (https://github.com/ajz2000/scikit-learn/), and then run the testCountVectorizer.py script. This should verify that our bugfix both passes the built-in scikit-learn unit tests, as well as the test suite that we have generated.

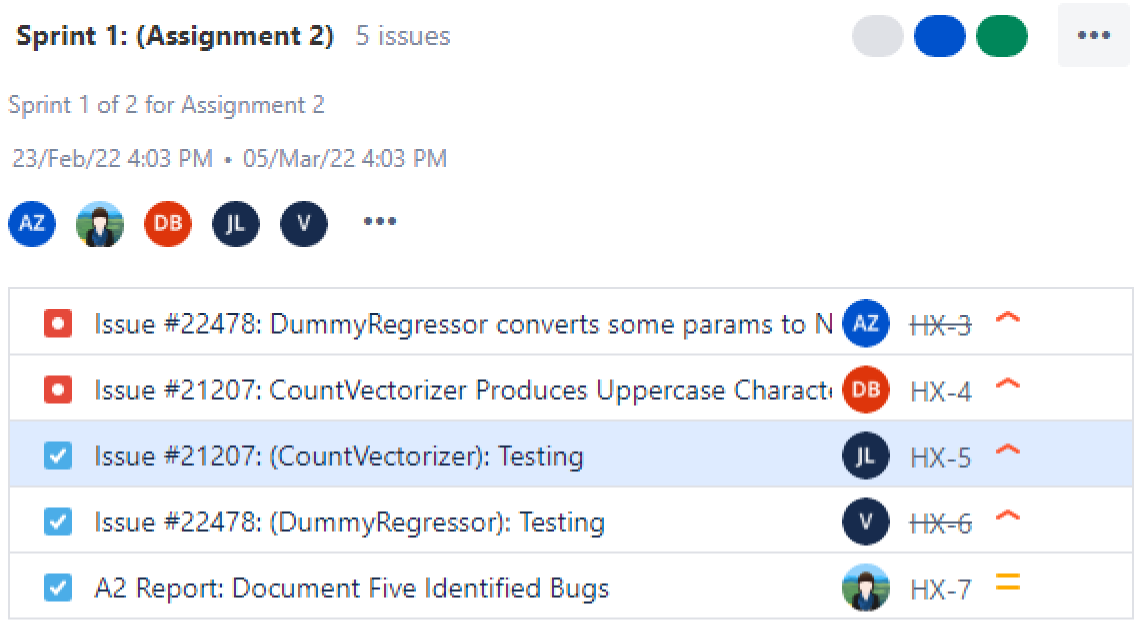
Development Process

Planning and Meetings

During this phase of development, our team's development process was largely structured around group sprint and goal-setting meetings in which tickets were created on Jira, and then assigned to smaller subgroups within the team. These goal-setting sessions were then followed by individual and/or small group work days guided by our Jira tickets, for which assigned team members would provide status updates during each subsequent standup meeting. This both ensured that development proceeded according to our plans, and the entire team could be advised on the status of each ticket to allow for us to both resolve issues and accommodate for individual needs and availability.

Typically, meetings were held on Mondays, Wednesdays, and Fridays, as according to our team agreement, and lasted between 20 and 40 minutes. These meetings consisted of first discussing whether and how we had accomplished the goals from the previous meeting, then briefly setting goals for the next meeting. Afterwards, if group members needed to talk to one another individually, they would break off into smaller groups to continue the conversation or collaborate on the development of an individual component.

The first Wednesday meeting of the assignment was reserved for sprint planning. The chosen bugs were broken down into tasks and subtasks (typically a development task and a testing task) on Jira beforehand, then the team was divided into two groups of three - one working on each bug. During the meeting, team members volunteered for the tasks they would complete, and appropriate tasks were moved across our Jira board to communicate progress on each respective task.



Sprint 1: (Assignment 2) 5 issues

Sprint 1 of 2 for Assignment 2

23/Feb/22 4:03 PM • 05/Mar/22 4:03 PM

AZ DB JL V •••

●	Issue #22478: DummyRegressor converts some params to N	AZ	~~HX-3~~	^
●	Issue #21207: CountVectorizer Produces Uppercase Characte	DB	HX-4	^
☑	Issue #21207: (CountVectorizer): Testing	JL	HX-5	^
☑	Issue #22478: (DummyRegressor): Testing	V	~~HX-6~~	^
☑	A2 Report: Document Five Identified Bugs		HX-7	=

Challenges

Overall, the development process was rather smooth, especially with fixing the bugs themselves, however the team initially faced a number of challenges while trying to understand the implementation details for our acceptance tests.

Another challenge the team faced was that this particular sprint coincided with a number of other commitments for many of our team members, such as midterm exams, other course assignments, and job interviews. As a result of these availability constraints, members of the team took extra time and attention to be accommodating of each other’s needs, and worked diligently within our available windows to ensure that our goals were still met in a timely manner.

Strategies

We initially spent considerable effort investigating/deciding which framework was best suited for our use case, what coverage would be sufficient, and what our acceptance conditions would be. To remedy this, we developed a standardised test template to base our cases around, and using this template we were able to fill in individual test case details and compare them against standard acceptance conditions.

To resolve scheduling constraints, the team worked together in two subgroups, with each focusing on fixing and testing a single bug. This allowed us to fix both bugs in parallel while better accommodating for individual teammates’ availability. These two subgroups shared status updates during standup meetings, and used Jira to communicate their state of development as to keep each other apprised throughout the development process.

Finally, as per our team agreement, our repository was managed using the Gitflow strategy despite the simplicity of these particular fixes.