TF-IDF using sklearn

Now we will see how we can implement this using sklearn in Python.
First, we will import `TfidfVectorizer` from `sklearn.feature_extraction.text`:

```
In [1]: from sklearn.feature_extraction.text import TfidfVectorizer
```

Now we will initialise the `vectorizer` and then call fit and transform over it to calculate the TF-IDF score for the text.

```
In [3]: vectorizer = TfidfVectorizer()
        response = vectorizer.fit_transform([S1, S2])
```

Under the hood, the sklearn fit_transform executes the following `fit` and `transform` functions. These can be found in the official sklearn library at GitHub.

```
def fit(self, X,
y=None):
                                """Learn the idf vector (global term weights)
                                Parameters
                                ----------
                                X : sparse matrix, [n_samples, n_features]
                                    a matrix of term/token counts
                                """
                                if not sp.issparse(X):
                                    X = sp.csc_matrix(X)
                                if self.use_idf:
                                    n_samples, n_features = X.shape
                                    df = _document_frequency(X)


                                    # perform idf smoothing if required
                                    df += int(self.smooth_idf)
                                    n_samples += int(self.smooth_idf)
# log+1 instead of
log makes sure
terms with zero
idf don't get

                               # suppressed entirely.
                               idf = np.log(float(n_samples) / df) + 1.0
                               self._idf_diag = sp.spdiags(idf, diags=0, m=n_features,
                                                           n=n_features, format='csr')
```

```
        return self


    def transform(self, X, copy=True):
        """Transform a count matrix to a tf or tf-idf representation
        Parameters
        ----------
        X : sparse matrix, [n_samples, n_features]
            a matrix of term/token counts
        copy : boolean, default True
            Whether to copy X and operate on the copy or perform in-
place
            operations.
        Returns
        -------
        vectors : sparse matrix, [n_samples, n_features]
        """
        if hasattr(X, 'dtype') and np.issubdtype(X.dtype, np.floating):
            # preserve float family dtype
            X = sp.csr_matrix(X, copy=copy)
        else:
            # convert counts or binary occurrences to floats
            X = sp.csr_matrix(X, dtype=np.float64, copy=copy)


        n_samples, n_features = X.shape


        if self.sublinear_tf:
            np.log(X.data, X.data)
            X.data += 1


        if self.use_idf:
            check_is_fitted(self, '_idf_diag', 'idf vector is not
fitted')


            expected_n_features = self._idf_diag.shape[0]
            if n_features != expected_n_features:
                raise ValueError("Input has n_features=%d while the
model"
                                 " has been trained with n_features=%d"
% (
                                     n_features, expected_n_features))
            # *= doesn't work
```

```
X = X * self._idf_diag

if self.norm:

X = normalize(X, norm=self.norm, copy=False)
return X
```

One thing to notice in the above code is that, instead of just the log of n_samples, 1 has been added to n_samples to calculate the IDF score. This ensures that the words with an IDF score of zero don't get suppressed entirely.

The output obtained is in the form of a skewed matrix, which is normalised to get the following result.

```
In [14]: print(response)

  (0, 6)	0.604379551537
  (0, 0)	0.42471718587
  (0, 3)	0.302189775769
  (0, 1)	0.302189775769
  (0, 4)	0.302189775769
  (0, 5)	0.42471718587
  (1, 6)	0.604379551537
  (1, 3)	0.302189775769
  (1, 1)	0.302189775769
  (1, 4)	0.302189775769
  (1, 7)	0.42471718587
  (1, 2)	0.42471718587
```

Thus we saw how we can easily code TF-IDF in just 4 lines using sklearn. Now we understand how powerful TF-IDF is as a tool to process textual data out of a corpus.