Abstract clustering

Clustering algorithm used to group abstract texts from research papers

Algorithm description

1. **Data Downloading and Extraction**: Automates data fetching and extraction from a specified source. `https://www.nsf.gov/awardsearch/download?DownloadFileName=2020&All=true`

2. **Preprocessing**: Cleans and preprocesses the text data for clustering by removing english stop words, custom stop words based on the most common used words in the abstracts, and a common found phrase.

3. **Clustering**: Uses K-Means to cluster abstracts based on TF-IDF and dimensionality-reduction techniques.

4. **Solution Builder**: That extracts the top words used per cluster to identify the possible topics of the clustered papers.

Preprocessing the abstracts

```
def preprocess_text(self, text: str) -> str:
    """..."""
    text = re.sub(pattern=r'<br\s*/?>', repl=' ', string=text)  # Remove HTML line breaks
    text = re.sub(pattern=r'&lt;br/&gt;', repl=' ', string=text)  # Handle HTML encoded breaks
    text = re.sub(pattern=self.phrase_to_remove, repl=' ', string=text)  # Remove the predefined phrase

    # Tokenize, lowercase, remove stopwords and lemmatize
    word_tokens = word_tokenize(text.lower())
    filtered_before_lemma = [w for w in word_tokens if w.isalpha() and w not in self.set_stopwords]
    lemmatized_text = [self.lemmatizer.lemmatize(token) for token in filtered_before_lemma]
    filtered_after_lemma = [w for w in lemmatized_text if w not in self.set_stopwords]

    return ' '.join(filtered_after_lemma)
```

Preprocessing the data for the clustering algorithm

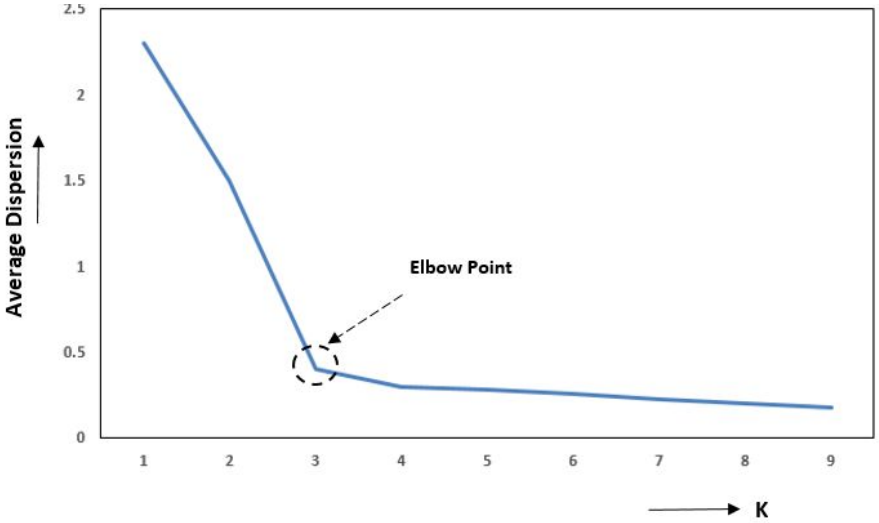
```
# Step 1: TF-IDF vectorization
tfidf_vectorizer = TfidfVectorizer(max_features=self.n_features_tfid)
tfidf_data = tfidf_vectorizer.fit_transform(preprocessed_abstracts)

# Step 2: Dimensionality reduction with TruncatedSVD and Normalization
svd = TruncatedSVD(n_components=self.n_components_truncatesvd)
normalizer = Normalizer(copy=False)
svd_and_normalizer = make_pipeline( *steps: svd, normalizer)
x_data = svd_and_normalizer.fit_transform(tfidf_data)

# Step 3: Get n-solutions based on the elbow_method and silhouette_score_for_range
kmeans_options = self._find_best_kmeans_k(x_data)
```

Selection of the k parameter in the k-means algorithm

Step 1) Find the Elbow point



Step 2) Do a local search to values near the elbow point based on Silhouette Score

```
k_min = max(elbow_point - self.n_solutions, 2)  # Ensure k_min is at least 2
k_max = elbow_point + self.n_solutions

# Dictionary to store model details for each k
kmeans_results = {}
# List to store silhouette scores for each k
silhouette_scores = []

for k in range(k_min, k_max + 1):
    # Apply KMeans
    kmeans = KMeans(n_clusters=k)
    cluster_labels = kmeans.fit_predict(data)

    # Calculate silhouette score
    score = silhouette_score(data, cluster_labels)
```

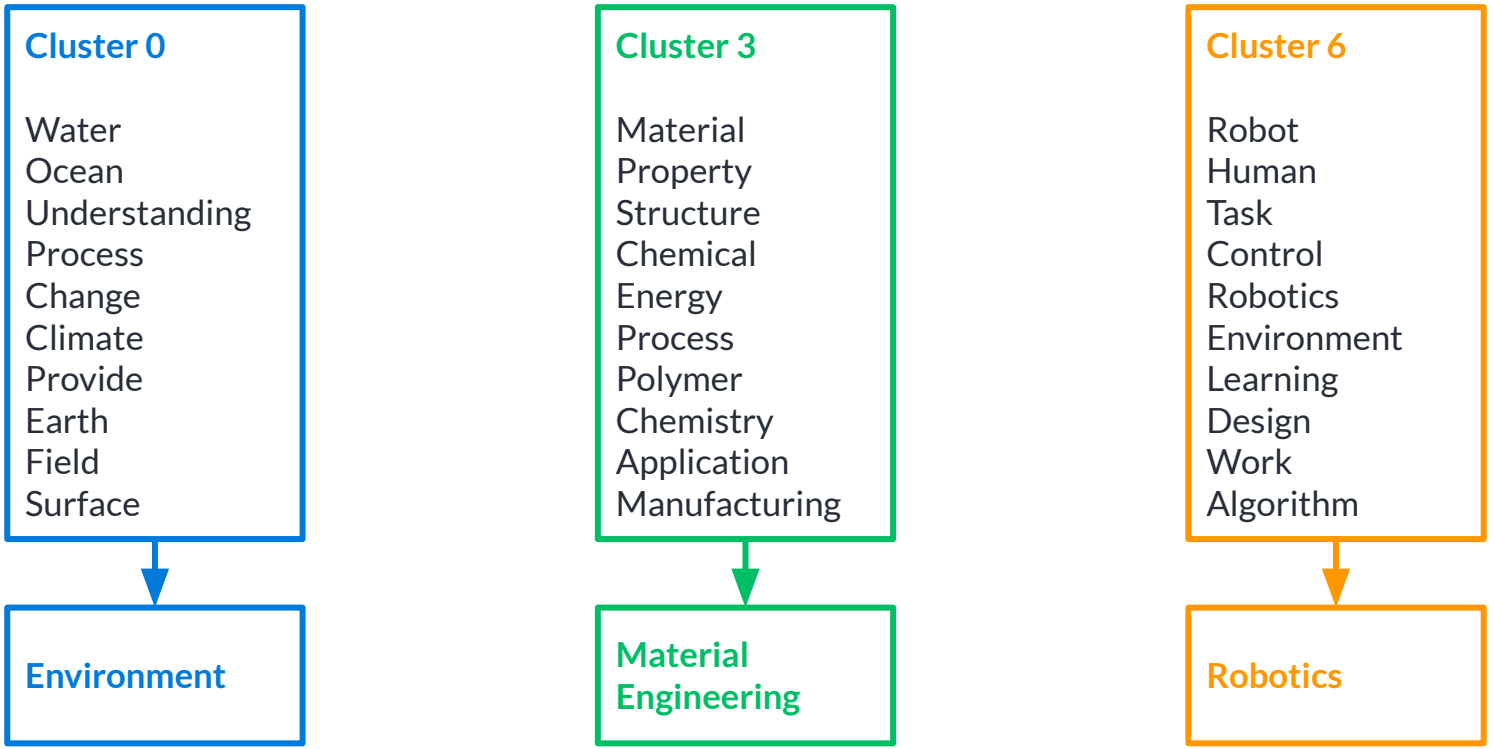
Solution Builder

```
for solution in solutions_found:
    solution.generate_pickle_file()
    solution.generate_graph()
    solution.generate_pdf()
```

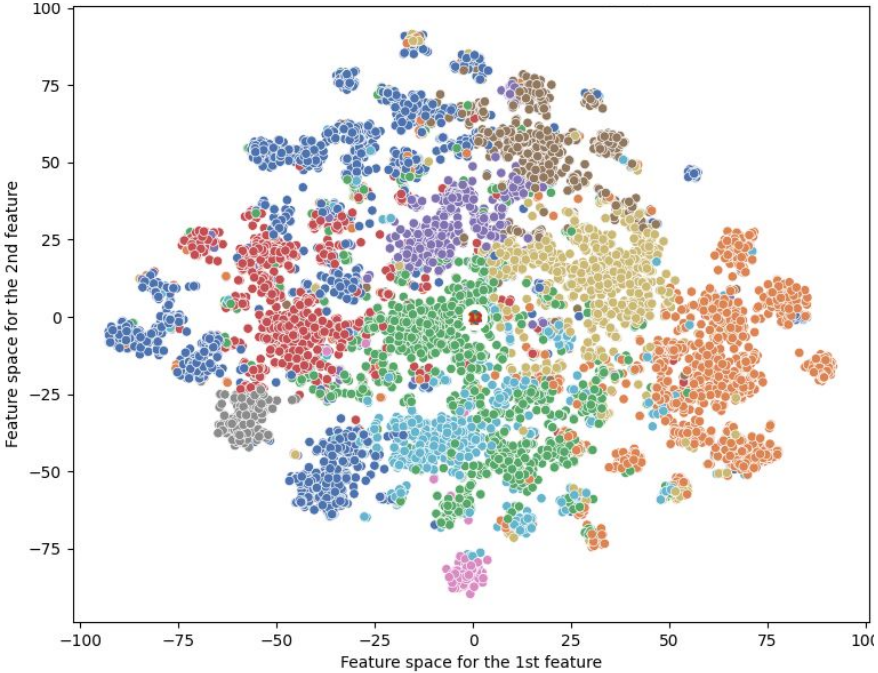
For each proposed solution the code saves:

1) Pickle File with the model information used in the cluster algorithm
2) A PNG File with a data (in a 2D representation)
3) A PDF file with the most common used words in each cluster for possible labeling

Examples of most used words in clusters and labeling



Cluster analysis for the 13-cluster solution



Cluster 0: Environment

Cluster 1: STEM Education

Cluster 2: Business Innovation

Cluster 3: Material Engineering

Cluster 4: Genetic Engineering

Cluster 5: Biodiversity

Cluster 6: Robotics

Cluster 7: Physics Engineering

Cluster 8: Health and Society

Cluster 9: Algorithms and Neural Networks

Cluster 10: Mathematics

Cluster 11: Engineering Studies

Cluster 12: Networks and Computation