Solutions of Ex.1

27.03.2017

Tasks

In the previous exercise you should have learned how to :

1. **Construct a classification pipeline**
2. Normalize text
3. Extract features from text
4. Experiment with classifiers
5. Evaluate results

Constructing a Pipeline : pipeline.py

1. Read the data
2. Split data for cross-validation
3. For each fold:
    - 3.1. Extract features
    - 3.2. Run Classifier
    - 3.3. Evaluate Classifier

1. Reading Data: get_dataset

1. Use FNC's implementation of load_dataset
2. Link Body Ids to Body text (in separate file)
3. Add dictionary, which will hold the features

Instance:

{'**Body ID**': '712',
 'Headline': "Police find mass graves with at least '15 bodies' near Mexico "
 'town where 43 students disappeared after police clash',
 'Stance': 'unrelated',
 'articleBody': 'Danny Boyle is directing the untitled film\n … later was responsible for creating the early Apple computers.',
 'features': {}
}

2. Cross-Validation: test_cross_validation

1. Get an instance of **KFold** from **sklearn.model_selection**
2. **Apply** k-fold over the dataset and **Iterate** over the folds
3. For each fold extract features, train classifier, predict labels, evaluate labels
4. Get average score for the folds

```
k_folds = KFold(n_splits=2, random_state=42)

for train_index, test_index in k_folds.split(X):
        X_train = [X[i] for i in train_index]
        X_test = [X[i] for i in test_index]

        predicted_labels =
run_classifier(train=X_train, test=X_test)

        score = get_fnc_score(X_test,
predicted_labels)
```

3. For each fold: run_classifier + get_fnc_score

1. Use sklearn's **Pipeline** to extract all features (put them in the features dictionary of each instance).
2. Include the **classifier** in the pipeline or run it separately
3. **Predict** the classes for the test data, running it over the same trained Pipeline

```
pipeline = Pipeline([
('preprocess_lemmas', TokenizedLemmas()),
('sent_len', SentenceLength()),
('tfidf', BagOfTfIDF(train)),
('pos', POS()),
('ner', NER()),
('word_overlap', WordOverlap()),
('transform', ToMatrix()),
('norm', MinMaxScaler()),
('clf', SVC())])

pipeline.fit(train, true_labels)

pipeline.predict(test)
```

Tasks

In the previous exercise you should have learned how to :

1. Construct a classification pipeline
2. **Normalize text**
3. Extract features from text
4. Experiment with classifiers
5. Evaluate results

Normalize Text: preprocess.py

TokenizedLemmas: get list of tokenized lemmas in body and headline. This can be later, in the pipeline, reused by the various features.

Different Normalizations of Text:

- **_remove_stopwords**
- **_normalize_word**
- **_get_tokenized_lemmas**
- **_clean** - lowercasing, removing non-alphanumerics

Tasks

In the previous exercise you should have learned how to :

1. Construct a classification pipeline
2. Normalize text
3. **Extract features from text**
4. Experiment with classifiers
5. Evaluate results

3. Extract Features from text: features.py

```
class Feature(TransformerMixin):

    """Feature Interface."""

    def fit(self, X, y=None, **fit_params):

    return self
```

Features in the sklearn's Pipeline should implement **TransformerMixin**

2 abstract methods:

1. fit (train-not always needed => abstract class Feature)
2. transform (produce features)

3. Extract Features from text: features.py

1. Subclass Feature for the separate features and implement at least transform method.
2. Add each feature to the dictionary of features.
3. Use ToMatrix to turn dictionary to NxM matrix of features with N instances and M features.
4. Normalize features to the [0,1] scale.

Tasks

In the previous exercise you should have learned how to :

1. Construct a classification pipeline
2. Normalize text
3. Extract features from text
4. **Experiment with classifiers**
5. Evaluate results

4. Experiment with classifiers

class sklearn.svm.SVC(***C**=1.0*, ***kernel**='rbf'*, ***degree**=3*, ***gamma**='auto'*, *coef0=0.0*, *shrinking=True*, *probability=False*, *tol=0.001*, *cache_size=200*, ***class_weight**=None*, *verbose=False*, *max_iter=-1*, *decision_function_shape=None*, *random_state=None*)[source]¶

```
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                     'C': [1, 10, 100, 1000]},
                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]

clf = GridSearchCV(SVC(C=1), tuned_parameters, cv=5,

                   scoring='%s_macro' % score)

clf.fit(X_train, y_train)[source]¶
```

Tasks

In the previous exercise you should have learned how to :

1. Construct a classification pipeline
2. Normalize text
3. Extract features from text
4. Experiment with classifiers
5. **Evaluate results**

5. Evaluate Results

- get_fnc_score - modified accuracy score
- sklearn.metrics:
  - accuracy_score
  - precision_score
  - classification report
  - recall_score
  - f1_score
  - confision_matrix