UDACITY

PROJECT

Build a Sign Language Recognizer

A part of the Artificial Intelligence Nanodegree and Specializations Program

PROJECT REVIEW

CODE REVIEW 3

NOTES

▾ my_model_selectors.py 3

```python
import math
import statistics
import warnings

import numpy as np
from hmmlearn.hmm import GaussianHMM
from sklearn.model_selection import KFold
from asl_utils import combine_sequences


class ModelSelector(object):
    '''
    base class for model selection (strategy design pattern)
    '''

    def __init__(self, all_word_sequences: dict, all_word_Xlengths: dict, this_word: str,
                 n_constant=3,
                 min_n_components=2, max_n_components=10,
                 random_state=14, verbose=False):
        self.words = all_word_sequences
        self.hwords = all_word_Xlengths
        self.sequences = all_word_sequences[this_word]
        self.X, self.lengths = all_word_Xlengths[this_word]
        self.this_word = this_word
        self.n_constant = n_constant
        self.min_n_components = min_n_components
        self.max_n_components = max_n_components
        self.random_state = random_state
        self.verbose = verbose

    def select(self):
        raise NotImplementedError

    def base_model(self, num_states):
        # with warnings.catch_warnings():
        warnings.filterwarnings("ignore", category=DeprecationWarning)
        # warnings.filterwarnings("ignore", category=RuntimeWarning)
        try:
            hmm_model = GaussianHMM(n_components=num_states, covariance_type="diag", n_iter=1000,
                                    random_state=self.random_state, verbose=False).fit(self.X, self.lengths)
            if self.verbose:
                print("model created for {} with {} states".format(self.this_word, num_states))
            return hmm_model
        except:
            if self.verbose:
                print("failure on {} with {} states".format(self.this_word, num_states))
            return None


class SelectorConstant(ModelSelector):
    """ select the model with value self.n_constant

    """

    def select(self):
        """ select based on n_constant value

        :return: GaussianHMM object
        """
        best_num_components = self.n_constant
        return self.base_model(best_num_components)


class SelectorBIC(ModelSelector):
    """ select the model with the lowest Bayesian Information Criterion(BIC) score

    http://www2.imm.dtu.dk/courses/02433/doc/ch6_slides.pdf
    Bayesian information criteria: BIC = -2 * logL + p * logN
    """

    def select(self):
        """ select the best model for self.this_word based on
        BIC score for n between self.min_n_components and self.max_n_components

        :return: GaussianHMM object
        """
        warnings.filterwarnings("ignore", category=DeprecationWarning)

        # TODO implement model selection based on BIC scores
        lowest_BIC = float('inf')
        best_model = None
        for i in range(self.min_n_components, self.max_n_components + 1):
            try:
                model = self.base_model(i)
                logL = model.score(self.X, self.lengths)
                logN = np.log(len(self.X))
                p = i ** 2 + 2 * model.n_features * i - 1
                #Bayesian Information Criteria （BIC）
```

AWESOME

The no. of free parameters has been calculated correctly. This reflects your understanding of the topic and the research done 👍
Formula is perfectly implemented!

```python
                BIC = -2 * logL + p * logN
                #select model with lowest BIC score
                if BIC < lowest_BIC:
                    lowest_BIC = BIC
                    best_model = model
            except:
                continue
        return best_model if best_model else self.base_model(self.n_constant)


class SelectorDIC(ModelSelector):
    ''' select best model based on Discriminative Information Criterion

    Biem, Alain. "A model selection criterion for classification: Application to hmm topology optimization."
    Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on. IEEE, 2003.
    http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.58.6208&rep=rep1&type=pdf
    https://pdfs.semanticscholar.org/ed3d/7c4a5f607201f3848d4c02dd9ba17c791fc2.pdf
    DIC = log(P(X(i))) - alpha/(M-1)SUM(log(P(X(all but i))) with the regularizer alpha is taken to be 1
    '''

    def select(self):
        warnings.filterwarnings("ignore", category=DeprecationWarning)

        # TODO implement model selection based on DIC scores
        highest_DIC = float('-inf')
        best_model = None
        M = self.max_n_components - self.min_n_components
        for i in range(self.min_n_components, self.max_n_components + 1):
            try:
                model = self.base_model(i)
                #collect logP(Xj|Til,thetail) over j!=i as scores
                scores = [model.score(*self.hwords[word]) for word in self.hwords.keys() if word != self.this_word]
                #Discriminative Information Criterion (DIC)
                DIC = model.score(self.X, self.lengths) - np.sum(scores)/(M-1)
```

AWESOME

Very impressive! The score "P(X(all but i))" was a bit difficult to calculate and you nailed it 👍
Formula for DIC has been perfectly implemented!

```python
                #select model with highest DIC score
                if DIC > highest_DIC:
                    highest_DIC = DIC
                    best_model = model
            except:
                continue
        return best_model if best_model else self.base_model(self.n_constant)



class SelectorCV(ModelSelector):
    ''' select best model based on average log Likelihood of cross-validation folds
    '''
    def select(self):
        warnings.filterwarnings("ignore", category=DeprecationWarning)

        # TODO implement model selection using CV
        highest_cv = float('-inf')
        best_model = None
        for i in range(self.min_n_components, self.max_n_components + 1):
            try:
                model = self.base_model(i)
                scores = []
                #select 2fold cross validation
                for train, test in KFold(n_splits=2)(self.sequences):
                    self.X, self.lengths = combine_sequences(train, self.sequences)
```

AWESOME

Good use of combine_sequences utility 👍

```
                    train_model = self.base_model(i)
                    X, lengths = combine_sequences(test, self.sequences)
                    scores.append(train_model.score(X, lengths))
                cv = np.mean(scores)
                #select model with highest CV score
                if cv > highest_cv:
                    highest_cv = cv
                    best_model = model
            except:
                continue
        return best_model if best_model else self.base_model(self.n_constant)
```

▸ asl_recognizer.html

▸ my_recognizer.py

RETURN TO PATH

Rate this review

Student FAQ