```
import numpy as np
import random
import warnings
from numpy.random import RandomState


class MAPEstimator():
    """
    Maximum A-Posteriori Estimator for musk probabilities

    Attributes
    ----------
    w_D   : D-dimensional vector of reals
            Defines weight vector
    prior : string
            Defines prior that will be used options: ('trivial','sas')
            'sas' stands for 'spike-and-slab'
    alpha : float, must be greater than 0
            Defines precision parameter of the multivariate Gaussian prior for the weight vector or the "spike" mul
    beta  : float, must be greater than 0 (optional)
            Defines precision parameter of the "slab" Gaussian on the GMM prior for the weight vector
    max_iter: integer
            Defines max number of iterations model can take to converge on fit()
    step_size: float must be greater than 0
            The step size of the gradient descent algorithm
    max_iter: int greater than 0
            Maximum number of iterations that the gradient descent algorithm will take


    Examples
    # TODO : Update this example or delete it
    --------
    >>> word_list = ['dinosaur', 'trex', 'dinosaur', 'stegosaurus']
    >>> mapEst = MAPEstimator(Vocabulary(word_list), alpha=2.0)
    >>> mapEst.fit(word_list)
    >>> np.allclose(mapEst.predict_proba('dinosaur'), 3.0/7.0)
    True

    >>> mapEst.predict_proba('never_seen-before')
    Traceback (most recent call last):
    ...
    KeyError: 'Word never_seen-before not in the vocabulary'
    """

    def __init__(self, w_D, prior='trivial', step_size_type = 'universal', alpha=1.0, beta=None, max_iter=30000, to
        #TODO decide on max_iter
        self.w_D = w_D
        self.c = 0
        self.alpha = float(alpha)
        self.max_iter = max_iter
        self.tol = tol
        self.step_size = step_size
        self.step_size_type = step_size_type
        self.iteration_count = iteration_count
        self.prior = prior
        self.loss_array = []
        if prior != 'trivial':
            self.beta = beta


    def fit(self, train_X, train_y):
        '''
        Fit this estimator to provided training data with first order stochastic gradient descent

        Args
        ----
        train_X : NxD array
            Each entry is a D-dimensional vector representing a training example
        train_y : Nx1 array
            Each entry is corresponding output class for training data


        Returns
        -------
        None. Internal attributes updated.

        Post Condition
        ---------------
        Attributes will updated based on provided word list
        * The 1D array count_V is set to the count of each vocabulary word
        * The integer total_count is set to the total length of the word list
        '''
        self.iteration_count = 0

        example_num = 0
        num_examples = len(train_X)

        self.loss_array = []

        if self.prior == 'trivial':

            count_10 = 0
            loss_10 = []
            L_avg = np.inf
            diff_L = np.inf
            L_avg_prev = np.inf

#             while(self.iteration_count <= self.max_iter) and (diff_L > self.tol):
#             # TODO : spit out a warning if the max_iter is reached
#                 #print('iteration: %i' % self.iteration_count)

#                 h_x = np.matmul(self.w_D.transpose(), train_X) + self.c
#                 #print('h_x: ' + str(h_x))
# #                 print('w_D: ' + str(self.w_D))
# #                 print('c: ' + str(self.c))
#                 sig = 1 / (1 + np.exp(-h_x))
# #                 print('sig: ' + str(sig))
# #                 print('train_y: ' + str(train_y[example_num]))
#                 L = np.dot(train_y, np.log(sig)) + np.dot((1-train_y), np.log(1-sig))
#                 print('L:' + str(L))
# #                 loss_10.append(L)
# #                 count_10 += 1

# #                 if count_10 == 1000:
#                 diff_L = np.abs(L_avg_prev - L)
#                 L_avg_prev = L

#                 self.w_D = self.w_D - self.step_size * (np.matmul(train_X.transpose(),(sig - train_y)) + self.alp
#                 self.c = self.c - self.step_size * (sig - train_y)

# #                 if example_num >= num_examples:
# #                     example_num = 0
# #                 self.iteration_count += 1

#             if self.iteration_count == self.max_iter:
#                 warnings.warn("Maximum iterations reached")

            prng = RandomState(136)

            while(self.iteration_count <= self.max_iter) and (diff_L > self.tol):
            # TODO : spit out a warning if the max_iter is reached
                #print('iteration: %i' % self.iteration_count)
                example_num = prng.randint(0, num_examples-1)
                h_x = np.dot(self.w_D, train_X[example_num]) + self.c
                #print('h_x: ' + str(h_x))
#                 print('w_D: ' + str(self.w_D))
#                 print('c: ' + str(self.c))
                sig = 1 / (1 + np.exp(-h_x))
#                 print('sig: ' + str(sig))
#                 print('train_y: ' + str(train_y[example_num]))
                L = train_y[example_num] * np.log(sig) + (1-train_y[example_num]) * np.log(1-sig)
                #print('L:' + str(L))
                loss_10.append(L)
                count_10 += 1

                if count_10 == 100000:
                    L_avg = np.mean(loss_10)
#                     print(L_avg)
                    self.loss_array.append(L_avg)
                    diff_L = np.abs(L_avg_prev - L_avg)
                    count_10 = 0
                    loss_10 = []
                    L_avg_prev = L_avg
                if self.step_size_type == 'universal':
                    self.w_D = self.w_D - self.step_size * ((sig - train_y[example_num]) * train_X[example_num] + s
                    self.c = self.c - self.step_size * (sig - train_y[example_num])
                else:
                    self.w_D = self.w_D - (self.step_size + train_y[example_num]*0.1)* ((sig - train_y[example_num]
                    self.c = self.c - (self.step_size + train_y[example_num]*0.1) * (sig - train_y[example_num])

#                 example_num += 1
#                 if example_num >= num_examples:
#                     example_num = 0
                self.iteration_count += 1

            if self.iteration_count == self.max_iter:
                warnings.warn("Maximum iterations reached")

        #else:
            #TODO: Code up the SGD for the sas prior

    def predict_proba(self, test_X):
        '''
        Predict probability of a given set of feature vectors under this model

        Args
        ----
        test_X : NxD vector
            Examples for which probability will be predicted

        Returns
        -------
        proba : float between 0 and 1

        Throws
        ------
        ValueError if hyperparameters do not allow MAP estimation
        KeyError if the provided word is not in the vocabulary
        '''

        lin_preds = np.matmul(test_X, self.w_D)

        sig_preds = 1 / (1 + np.exp(-lin_preds))

        return sig_preds

    def score(self, test_X, test_y):
        ''' Compute the average log probability of words in provided list

        Args
        ----
        test_X : NxD array
            Each entry is a D-dimensional vector representing a test example
        test_y : Nx1 array
            Each entry is corresponding output class for test data

        Returns
        -------
        avg_log_proba : float between (-np.inf, 0.0)
        '''
        correct_count = 0
        num_examples = len(test_X)

        pred_proba = self.predict_proba(test_X)
        pred_class = pred_proba > 0.5
        is_correct = pred_class == test_y

        return np.sum(is_correct) / num_examples
```