```python

import numpy as np
import re
from gensim.test.utils import get_tmpfile
from gensim.models import KeyedVectors
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import NearestNeighbors
from sklearn.svm import LinearSVC
from sklearn.mixture import GaussianMixture

regex_punct = re.compile(r"[^ A-Za-z]+")
regex_proper_noun = re.compile(r"([^?.!])(\s+)([A-Z]\w*)")
regex_spaces = re.compile(r"\s+")

def check(string):
    return re.sub(regex_proper_noun, r"\1", string)

def fix_str(string):
    return re.sub(regex_spaces, " ", re.sub(regex_punct, "", re.sub(regex_proper_noun, r"\1", string)).lower())

def clean_text(filepath):
    f = open(filepath, "r")
    ctr = 0

    NUM_GOOD = None
    NUM_BAD = None

    good_strings = []
    bad_strings = []

    #doesn't matter, small loop.

    for line in f:
        if(ctr == 0):
            nums = line.split(' ')
            NUM_GOOD = int(nums[0])
            NUM_BAD = int(nums[1])
        elif(ctr < NUM_GOOD):
            good_strings.append(fix_str(line))
        else:
            bad_strings.append(fix_str(line))
        ctr += 1

    vectorizer = TfidfVectorizer()
    X = vectorizer.fit_transform(good_strings+bad_strings)
    sums = X.sum(axis=1)

    #since the dataset is quite small, I can get away with for loops instead of vectorization.

    model = KeyedVectors.load(get_tmpfile("glove100"), mmap='r')
    dim = model['dog'].shape[0]
    words = vectorizer.get_feature_names()
    X_train = np.zeros((X.shape[0],dim))
    print(dim)

    for doc_idx in range(X.shape[0]):
        weighted_vect = np.zeros(dim)
        for word_idx in range(X.shape[1]):
            wvec = None
            try:
                wvec = model[words[word_idx]]
            except KeyError:
                wvec = np.zeros(dim)
            weighted_vect += X[doc_idx,word_idx]*wvec
        weighted_vect /= sums[doc_idx,0]
        X_train[doc_idx] = weighted_vect.T
    f.close()
    return X_train, NUM_GOOD

def _knn(nn_ct = 3):
    X_train, train_bdry = clean_text("ggold.txt")
    X_test, test_bdry = clean_text("validation_set.txt")
    model = NearestNeighbors(n_neighbors=nn_ct, algorithm='ball_tree').fit(X_train)
    #print(X_test)
    ids = model.kneighbors(X_test, return_distance=False)
    check = np.zeros(X_test.shape[0],dtype=bool)
    check[:test_bdry] = True
    return np.sum((np.sum(ids < train_bdry, axis=1) > nn_ct//2) == check)/X_test.shape[0]

def _svm(gamma=100, tolerance=1e-4, iter_ct=1000):

    X_train, train_bdry = clean_text("ggold.txt")
    X_test, test_bdry = clean_text("ggold.txt")

    train_labels = np.zeros(X_train.shape[0], dtype=bool)
    train_labels[:train_bdry] = True

    model = LinearSVC(random_state=0, max_iter=iter_ct, C=1/gamma, tol=tolerance)
    model.fit(X_train, train_labels)

    gen_labels = model.predict(X_test)
    check = np.zeros(X_test.shape[0],dtype=bool)
    check[:test_bdry] = True

    print(check == gen_labels)

    return np.sum(check == gen_labels)/X_test.shape[0]

```