SVM

March 12, 2023

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from google.colab import drive
from sklearn.feature_extraction import text
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
import random
import time
import re
import string
from sklearn.naive_bayes import GaussianNB, MultinomialNB
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, chi2,␣
 ↪f_classif,mutual_info_classif,f_regression
from sklearn.preprocessing import Normalizer
from sklearn import model_selection
from sklearn import svm
import nltk
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk import word_tokenize
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet
from nltk.tokenize.treebank import TreebankWordDetokenizer
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
nltk.download('omw-1.4')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
```

```
nltk.download('wordnet')
nltk.download('stopwords')
from sklearn.svm import SVC
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data…
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[2]:
```
#import the data
drive.mount('/content/gdrive/', force_remount=True)

train_data_initial = pd.read_csv('/content/gdrive/MyDrive/ecse551-mp2/train.
 ↪csv')
test_data = pd.read_csv('/content/gdrive/MyDrive/ecse551-mp2/test.csv')

print('shape train:',train_data_initial.shape)
print('shape test:',test_data.shape)
```

```
Mounted at /content/gdrive/
shape train: (718, 2)
shape test: (279, 2)
```

[3]:
```
def shuffle_data(df):
    random.seed(0)  # Use a fixed seed for the random number generator
    df = df.sample(frac=1, random_state=0).reset_index(drop=True)
    return df
```

[4]:
```
#function for creating the test csv file to upload to kaggle
def create_test_csv(data, outfile_name):
  rawdata= {'subreddit':data}
  csv = pd.DataFrame(rawdata, columns = ['subreddit'])
  csv.to_csv(outfile_name,index=True, header=True)
  print ("File saved.")
```

[9]:
```
#shuffle the data and split the features from the label
train_data = shuffle_data(train_data_initial)

train_x = train_data["body"]
train_y = train_data["subreddit"]
test_x = test_data["body"]
```

[6]:
```
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'\d+', '', text)
    return text
```

[7]:
```
#create a dictionary of stop words
stop_words_nltk = set(stopwords.words('english'))
stop_words_sklearn = text.ENGLISH_STOP_WORDS
stop_words_library = stop_words_sklearn.union(stop_words_nltk)
```

[ ]:
```
############################################################
```

[11]:
```
#initial training without removing parameters
t_start = time.time()

pipe_params = {
    'classify__C': [0.1, 1, 10],
    'classify__kernel': ['linear', 'rbf']
}

vectorizer = CountVectorizer()
model = SVC()

pipe = Pipeline(
    [("vect", vectorizer),("classify",model)]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
The best accuracy is 90.251.
The winning parameters are {'classify__C': 10, 'classify__kernel': 'rbf'}
Run time: 15.285731792449951 seconds
```

[13]:
```
#testing stop words
t_start = time.time()

pipe_params = {
    "vect__binary": [False,True],
    "vect__stop_words":␣
 ↪[list(stop_words_nltk),list(stop_words_sklearn),list(stop_words_library)],
    "selecter__k":[5000,6000,3000],
    "classify__alpha" : [0.001, 0.01, 0.1,0.02,0.5]
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)


pipe = Pipeline(
    [("vect", vectorizer), ("selecter", selecter),("classify", SVC())]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 90 candidates, totalling 450 fits
The best accuracy is 92.198.
The winning parameters are {'classify__alpha': 0.1, 'selecter__k': 5000,
'vect__binary': False, 'vect__stop_words': ['when', 'few', 'very', 'between',
'nine', 'd', 'elsewhere', 'ourselves', 'wherein', 'several', 'still', 'even',
'seeming', 'an', 'becoming', 'below', 'give', 'nobody', 'behind', 'thru',
'mustn', 'ma', 'about', 'if', 'must', 'toward', 'what', 'on', 'through', 'ever',
'anyhow', 'there', 'fill', 'empty', 'by', 'these', 'co', 'full', 'therefore',
"didn't", 'won', 'you', 'another', 'within', 'seemed', 'sometimes', 'doesn',
'meanwhile', 'becomes', 'thence', 'fifteen', 'take', 'to', 'will', 'hadn',
'found', 'have', 'four', 'them', 'whereby', 'were', 'theirs', 'be', "wasn't",
```

```
'six', 'nevertheless', 'formerly', 'are', 'although', 'cry', 'sometime', 'eg',
'further', 'perhaps', 'again', 'under', 'this', 'alone', 'us', 'might', 'see',
'do', 'both', 'against', 'con', 'having', 'since', 'around', 'needn', 'himself',
'system', 'among', 'eleven', 'for', 'former', 'it', 'onto', 'interest',
'anyway', 'hereby', 'out', 'after', 'itself', "don't", "isn't", 'before',
'besides', "should've", 'nothing', 'can', "you'll", 'or', 'get', 'anyone',
'move', 'whence', 'mine', 'a', 'nor', 'other', 'per', 'back', 'last',
'everywhere', 'm', 'part', 'own', 'name', 'should', 'y', 'was', 'didn',
'whereupon', 'mightn', 'over', 'haven', 'bottom', "hadn't", 'thereafter',
'anything', 'inc', 'above', 'de', 'noone', 'don', "it's", 'one', 'from',
'someone', 'during', 'therein', 'everyone', 'well', 'his', 'i', 'being',
'without', 'while', 'o', 'done', 'whatever', 'yet', "shouldn't", 'hence', 'go',
'hasn', 'afterwards', 'seems', 'as', "that'll", 'may', 'though', 'hereafter',
'however', 'made', 'seem', 'him', 'amoungst', 'somehow', 'mostly', 'whither',
'none', 'then', 'could', 'also', 'how', 't', 'off', 'others', 'ie', 'latter',
'serious', 'describe', 'everything', 'across', 'll', 'yourself', 'front',
'same', 'yours', 'next', 'no', 'else', 'via', 'thin', "wouldn't", 'side', 'up',
'every', 'two', 'mill', 'something', 'already', 'together', 'many', 'thus',
'but', 'that', 'rather', 'neither', 'nowhere', 'your', 'its', 'except', 'ten',
'keep', 'show', 'yourselves', 'my', "couldn't", 'where', 'much', 'he', 'herein',
'down', 'wherever', 'with', 'due', 'namely', 'please', 'always', 'did',
"you've", "shan't", 'into', 'cant', 'less', 'five', 'had', 'twelve', 'and',
'along', 'almost', "haven't", 'most', "aren't", 'third', 'some', 'hundred',
'they', 'such', 'been', 're', 'indeed', 'often', 'would', "mightn't", 'just',
'me', 'call', 'weren', 'now', 'of', 'throughout', 'thick', 'whenever', 'until',
'cannot', 'least', 'thereupon', 'beside', 'hers', "doesn't", 'beyond',
'thereby', 'towards', 'couldn', 'top', 'once', 'whole', 'three', 'couldnt',
'ours', 'has', 'more', 'forty', 'whereafter', 'amongst', 'beforehand', 's',
'became', 'fifty', 'wasn', "you'd", 'am', 'twenty', "needn't", 'each', 'does',
'in', 'otherwise', 'ain', 'bill', 'become', 'than', 'detail', 'at', 'put',
'themselves', 'because', 'shan', 'latterly', 'sixty', 'our', 'we', 'eight',
'not', 'amount', 'too', 'fire', 'whereas', 'who', 'doing', 'isn', 'whom', 'any',
'whether', "won't", 'un', 'etc', 'so', 'her', 'shouldn', "weren't", 'myself',
'upon', 'somewhere', 'never', 'which', 'aren', "you're", 'why', 'ltd',
"mustn't", 'hereupon', 'herself', 've', 'whose', 'is', "hasn't", 'enough',
'all', 'only', 'those', 'whoever', 'wouldn', 'anywhere', 'hasnt', 'the',
'their', 'sincere', "she's", 'here', 'either', 'first', 'find', 'moreover',
'she']}
Run time: 74.24770450592041 seconds
```

[ ]:
```
#testing normalizer
t_start = time.time()

pipe_params = {
    "vect__binary": [False,True],
    "vect__stop_words": [list(stop_words_library)],
    "selecter__k":[5000,3000],
```

```
    "classify__alpha" : [0.001, 0.01, 0.1,0.02,0.5],
    "normalizer__norm": ['l2','l1','max']
}

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
normalizer = Normalizer()

pipe = Pipeline(
    [("vect", vectorizer), ("selecter",␣
 ↪selecter),("normalizer",normalizer),("classify", SVC())]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)

print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits
The best accuracy is 90.53.
The winning parameters are {'classify__alpha': 0.02, 'normalizer__norm': 'l1',
'selecter__k': 5000, 'vect__binary': False, 'vect__stop_words':
frozenset({'only', 'whereby', 'thereby', 'within', 'that', 'top', 'bill',
'here', 'ain', 'anyway', 'himself', 'full', 'there', 'nine', 'well', 'couldn',
'would', 'they', "hadn't", 'along', 'whether', 'more', 'around', 'an', 'hasnt',
'she', 'never', 'be', 'already', 'de', 'else', 'whose', 'anyone', 'wasn',
'without', 'whole', 'thru', 'even', "it's", "doesn't", 'none', 'made', 'to',
'not', 'still', 'sometimes', 'my', 'yours', 'from', 'keep', 'who', "you'd",
'further', 'his', 'might', 'whoever', 'through', 'formerly', 'describe', 'ltd',
'whereafter', 'whenever', 'being', 'us', 'upon', 'ourselves', 'show', 'against',
'we', 'cannot', 'anyhow', 'doing', 'get', 'has', 'with', 'by', 'just',
'seeming', 'whatever', 'although', 'most', 'when', 'thereafter', 'below', 'and',
'yourself', 'how', 'down', 'back', 'five', 'four', "you'll", 'until', 'what',
'while', 'as', 'fill', 'those', "aren't", 'but', 'call', 'could', 'their',
'then', 'noone', 'six', 'which', 'ma', 'throughout', 'anything', 'part',
'itself', 'again', 'twelve', "won't", 'last', 'whereas', 'up', 'perhaps',
'ours', 'all', "needn't", 'been', 'eight', 'etc', 'via', 'amoungst', 'o',
'either', 'least', 'three', "couldn't", 'didn', 'beforehand', 'latter',
'thence', 'amongst', 'hereby', 'whither', 'became', 'couldnt', "wouldn't",
```

```
'onto', 'seem', 'm', 'almost', 'fifty', 'nowhere', 'anywhere', 'therein',
"didn't", 'under', 'was', 'others', "haven't", 'thin', 'behind', 'too', 'done',
'after', 'should', 'third', 'per', 'across', "you're", 'becoming', 'its',
'these', 'them', 've', 'enough', 'if', 'seems', 'beyond', 'fire', 'front',
'somehow', 'however', 'everything', 'a', 'empty', 'y', 'together', 'shouldn',
'for', 'hereupon', 'hadn', 'same', 'hence', 'indeed', 'over', 'no', 'doesn',
'have', 'forty', 'ten', 'amount', 'having', 'hasn', 'any', 'off', 'such',
'first', 'themselves', "don't", 'bottom', 'rather', "mustn't", 'do', 'are',
'can', 'besides', 'somewhere', 'fifteen', "weren't", 'since', 'also', 'mill',
'often', 'nobody', 'due', 'wherever', 'did', 'always', 'thereupon', "hasn't",
'name', 'therefore', 'un', 'go', 'aren', "isn't", 'were', 'out', 'sincere',
"mightn't", 'thick', 'inc', 'become', 'alone', 'several', 'this', 'he', 'among',
'll', 'detail', 'during', 'mostly', 'you', 'won', 'namely', 'our', 'yourselves',
'in', 'why', 'herein', 'wherein', 'serious', 'both', 'the', 'toward',
"shouldn't", 'on', 'another', 'because', 'haven', 'needn', 'please', 'next',
'find', 'your', 'moreover', "should've", 'though', 's', "wasn't", 'nothing',
'less', 'system', 'twenty', 'now', 'about', 'mustn', 'herself', 'hers', 'or',
'every', 'than', 'everywhere', "you've", 'latterly', "she's", 'afterwards',
'weren', 'above', 'side', 'everyone', 'eg', 'elsewhere', 're', 'hereafter',
'where', 'see', 'very', 'yet', 'myself', 'two', 'former', 'cry', 'towards',
'thus', 'i', 'd', 'ie', 'whence', 'con', 'move', 'mightn', 'am', 'don',
'hundred', 'of', 'whereupon', 'other', 'once', 'me', 'her', 'wouldn',
'otherwise', 'found', 'seemed', 'give', 'becomes', 'it', 'at', 'between',
'something', 'so', 'him', 'into', 'neither', 't', 'put', 'except', 'few',
'beside', 'whom', 'meanwhile', 'nevertheless', 'mine', 'isn', 'does', 'before',
'may', 'ever', 'theirs', 'will', 'eleven', "that'll", 'one', "shan't", 'take',
'sixty', 'sometime', 'each', 'had', 'interest', 'own', 'is', 'much', 'shan',
'cant', 'nor', 'co', 'many', 'must', 'some', 'someone'})}
Run time: 26.246994018554688 seconds
```

[14]:
```
#stem lemmatizer
def get_wordnet_pos(word):
    """Map POS tag to first character lemmatize() accepts"""
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ,
                "N": wordnet.NOUN,
                "V": wordnet.VERB,
                "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

class LemmaTokenizer_Pos:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos =get_wordnet_pos(t)) for t in␣
 ↪word_tokenize(doc) if t.isalpha()]
```

```
class LemmaTokenizer:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) if t.
 ↪isalpha()]

class LemmaTokenizer_word:
     def __init__(self):
       self.wnl = WordNetLemmatizer()
     def __call__(self, doc):
       return [self.wnl.lemmatize(t,pos ="v") for t in word_tokenize(doc) ]


class StemTokenizer:
     def __init__(self):
       self.wnl =PorterStemmer()
     def __call__(self, doc):
       return [self.wnl.stem(t) for t in word_tokenize(doc) if t.isalpha()]
```

[ ]:
```
#testing lemma
t_start = time.time()

pipe_params = {
    "vect__binary": [False,True],
    "vect__stop_words": [list(stop_words_library)],
    "vect__tokenizer": [LemmaTokenizer_word()],
    "selecter__k":[5000,3000]
    }

vectorizer = CountVectorizer()
selecter = SelectKBest(chi2)
normalizer = Normalizer()

pipe = Pipeline(
    [("vect", vectorizer), ("selecter",␣
 ↪selecter),("normalizer",normalizer),("classify", SVC())]
)

grid = model_selection.GridSearchCV(pipe, pipe_params, verbose=1, n_jobs=-1)

grid.fit(train_x, train_y)

t_end = time.time()

elapsed_time = t_end-t_start
accuracy = round(grid.best_score_ * 100,3)
```

```
print(f"The best accuracy is {accuracy}.")
print(f"The winning parameters are {grid.best_params_}")
print(f"Run time: {elapsed_time} seconds")
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits

/usr/local/lib/python3.8/dist-packages/sklearn/feature_extraction/text.py:396:
UserWarning: Your stop_words may be inconsistent with your preprocessing.
Tokenizing the stop words generated tokens ["'d", "'ll", "'re", "'s", "'ve",
'make', "n't", 'need', 'sha', 'win', 'wo'] not in stop_words.
  warnings.warn(

The best accuracy is 89.833.
The winning parameters are {'classify__alpha': 0.1, 'selecter__k': 5000,
'vect__binary': False, 'vect__stop_words': frozenset({'only', 'whereby',
'thereby', 'within', 'that', 'top', 'bill', 'here', 'ain', 'anyway', 'himself',
'full', 'there', 'nine', 'well', 'couldn', 'would', 'they', "hadn't", 'along',
'whether', 'more', 'around', 'an', 'hasnt', 'she', 'never', 'be', 'already',
'de', 'else', 'whose', 'anyone', 'wasn', 'without', 'whole', 'thru', 'even',
"it's", "doesn't", 'none', 'made', 'to', 'not', 'still', 'sometimes', 'my',
'yours', 'from', 'keep', 'who', "you'd", 'further', 'his', 'might', 'whoever',
'through', 'formerly', 'describe', 'ltd', 'whereafter', 'whenever', 'being',
'us', 'upon', 'ourselves', 'show', 'against', 'we', 'cannot', 'anyhow', 'doing',
'get', 'has', 'with', 'by', 'just', 'seeming', 'whatever', 'although', 'most',
'when', 'thereafter', 'below', 'and', 'yourself', 'how', 'down', 'back', 'five',
'four', "you'll", 'until', 'what', 'while', 'as', 'fill', 'those', "aren't",
'but', 'call', 'could', 'their', 'then', 'noone', 'six', 'which', 'ma',
'throughout', 'anything', 'part', 'itself', 'again', 'twelve', "won't", 'last',
'whereas', 'up', 'perhaps', 'ours', 'all', "needn't", 'been', 'eight', 'etc',
'via', 'amoungst', 'o', 'either', 'least', 'three', "couldn't", 'didn',
'beforehand', 'latter', 'thence', 'amongst', 'hereby', 'whither', 'became',
'couldnt', "wouldn't", 'onto', 'seem', 'm', 'almost', 'fifty', 'nowhere',
'anywhere', 'therein', "didn't", 'under', 'was', 'others', "haven't", 'thin',
'behind', 'too', 'done', 'after', 'should', 'third', 'per', 'across', "you're",
'becoming', 'its', 'these', 'them', 've', 'enough', 'if', 'seems', 'beyond',
'fire', 'front', 'somehow', 'however', 'everything', 'a', 'empty', 'y',
'together', 'shouldn', 'for', 'hereupon', 'hadn', 'same', 'hence', 'indeed',
'over', 'no', 'doesn', 'have', 'forty', 'ten', 'amount', 'having', 'hasn',
'any', 'off', 'such', 'first', 'themselves', "don't", 'bottom', 'rather',
"mustn't", 'do', 'are', 'can', 'besides', 'somewhere', 'fifteen', "weren't",
'since', 'also', 'mill', 'often', 'nobody', 'due', 'wherever', 'did', 'always',
'thereupon', "hasn't", 'name', 'therefore', 'un', 'go', 'aren', "isn't", 'were',
'out', 'sincere', "mightn't", 'thick', 'inc', 'become', 'alone', 'several',
'this', 'he', 'among', 'll', 'detail', 'during', 'mostly', 'you', 'won',
'namely', 'our', 'yourselves', 'in', 'why', 'herein', 'wherein', 'serious',
'both', 'the', 'toward', "shouldn't", 'on', 'another', 'because', 'haven',
'needn', 'please', 'next', 'find', 'your', 'moreover', "should've", 'though',
```

```
's', "wasn't", 'nothing', 'less', 'system', 'twenty', 'now', 'about', 'mustn',
'herself', 'hers', 'or', 'every', 'than', 'everywhere', "you've", 'latterly',
"she's", 'afterwards', 'weren', 'above', 'side', 'everyone', 'eg', 'elsewhere',
're', 'hereafter', 'where', 'see', 'very', 'yet', 'myself', 'two', 'former',
'cry', 'towards', 'thus', 'i', 'd', 'ie', 'whence', 'con', 'move', 'mightn',
'am', 'don', 'hundred', 'of', 'whereupon', 'other', 'once', 'me', 'her',
'wouldn', 'otherwise', 'found', 'seemed', 'give', 'becomes', 'it', 'at',
'between', 'something', 'so', 'him', 'into', 'neither', 't', 'put', 'except',
'few', 'beside', 'whom', 'meanwhile', 'nevertheless', 'mine', 'isn', 'does',
'before', 'may', 'ever', 'theirs', 'will', 'eleven', "that'll", 'one', "shan't",
'take', 'sixty', 'sometime', 'each', 'had', 'interest', 'own', 'is', 'much',
'shan', 'cant', 'nor', 'co', 'many', 'must', 'some', 'someone'}),
'vect__tokenizer': <__main__.LemmaTokenizer_word object at 0x7f4b6a36c940>}
Run time: 74.08984041213989 seconds
```

[ ]:
```
# Step 5: Make predictions on test data using the trained model
```

[ ]:
```
############################################################ final
```

[17]:
```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.svm import SVC
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
import numpy as np



# define the stop words
stop_words = set(stopwords.words('english'))

# define the pipeline
pipeline = Pipeline([
    ('vectorize', CountVectorizer(stop_words=list(stop_words),␣
 ↪binary=True,lowercase = False,preprocessor=preprocess_text)),
    ('selector', SelectKBest(chi2, k=3000)),
    ('clf', SVC())
])

cross_val_score = np.mean(model_selection.cross_val_score(pipeline, train_x,␣
 ↪train_y, cv=5,n_jobs=-1,verbose=1))
print('cross_val_score->',cross_val_score)

pipeline.fit(train_x, train_y)

test_x_processed = pipeline.named_steps['vectorize'].transform(test_x)
```

```
test_x_selected = pipeline.named_steps['selector'].transform(test_x_processed)



y_pred = pipeline.predict(test_x)

create_test_csv(y_pred,"SVM.csv")
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done   5 out of   5 | elapsed:    4.6s finished

cross_val_score-> 0.8551379176379175
File saved.
```

[ ]:

```

```

```
File saved.
```