Movie reviews classifier

Problem

Creation of a system that distinguishes reviews of movies among two classes: positive and negative.

Input

```
example0 = ['I like this movie :)']
example1 = ['I hate this movie']
example2 = ['I do not like it']
example3 = ['This is my new favourite movie now']
```

Software takes array of sentences as input.

Output

```
Text: ['This is my new favourite movie now']
Class: Positive
Probability: 68.06%
```

As output system return a class of review and probability of belonging to this class.

0 - positive
1 - negative

Dataset

As dataset Internet Movie Database was used. It contains 50,000 movie reviews. It can be download from http://ai.stanford.edu/~amaas/data/sentiment.

The dataset consist of:

Review	Class
After repeatedly saying how brilliant so…	1
This is a well-worn story about a man	0

Text preprocessing

```
def tokenizer(text):
    text = re.sub('<[^>]*>', ' ', text)
    emoticons = re.findall('(?::|;|=)(?:-)?(?:\)|\(|D|P)', text.lower())
    text = re.sub('[\W]+', ' ', text.lower()) + ' '.join(emoticons).replace('-', '')
    tokenized = [w for w in text.split() if w not in stop]
    return tokenized
```

Tokenizer function cleans up the raw text data and then extracts the word markers while removing the skipped words.

```
def stream_docs(path):
    with open(path, 'r', encoding='utf-8') as csv:
        next(csv)
        for line in csv:
            text, label = line[:-3], int(line[-2])
            yield text, label
```

Each time function return subsequent text.

```
def get_minibatch(doc_stream, size):
    docs, y = [], []
    try:
        for _ in range(size):
            text, label = next(doc_stream)
            docs.append(text)
            y.append(label)
    except StopIteration:
        return None, None
    return docs, y
```

get_minibatch function get text with given size from array generated by stream_docs

Training

To train model out-of-core learning technique was used. It allows incremental matching of the classifier using smaller subsets of the training examples.

```
vect = HashingVectorizer(decode_error='ignore', n_features=2 ** 21, preprocessor=None, tokenizer=tokenizer)
clf = SGDClassifier(loss='log', random_state=1, max_iter=1)
```

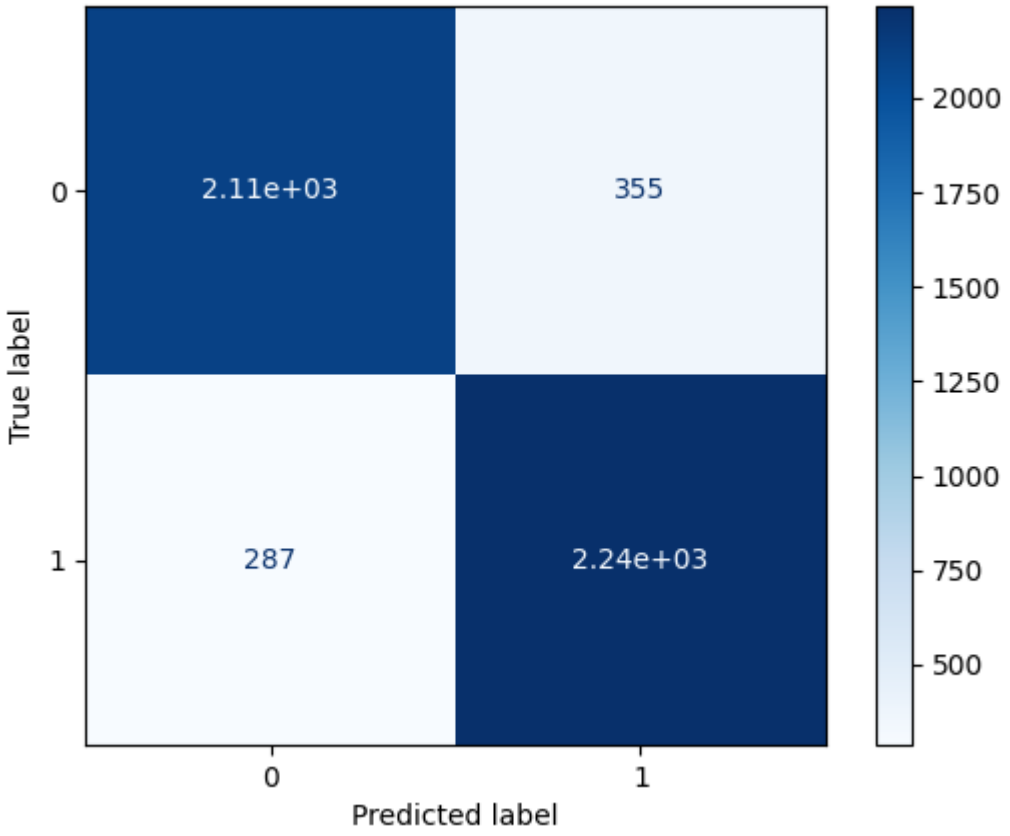
vect variable is assigned to class HashingVectorizer, where tokenizer function and number of features were defined. HashingVectorizer is a vectorizer which uses the hashing trick to find the token string name to feature integer index mapping. Conversion of text documents into matrix is done by this vectorizer where it turns the collection of documents into a sparse matrix which are holding the token occurence counts. SGDClassifier was used as train model. As loss parameter it takes “loss” which means that it becomes a logistic regression classifier.

```
for _ in range(45):
    X_train, y_train = get_minibatch(doc_stream, size=1000)
    if not X_train:
        break
    X_train = vect.transform(X_train)
    clf.partial_fit(X_train, y_train, classes=classes)
```

During each iteration another part of data is loaded and using partial fit we do not need to load whole data.

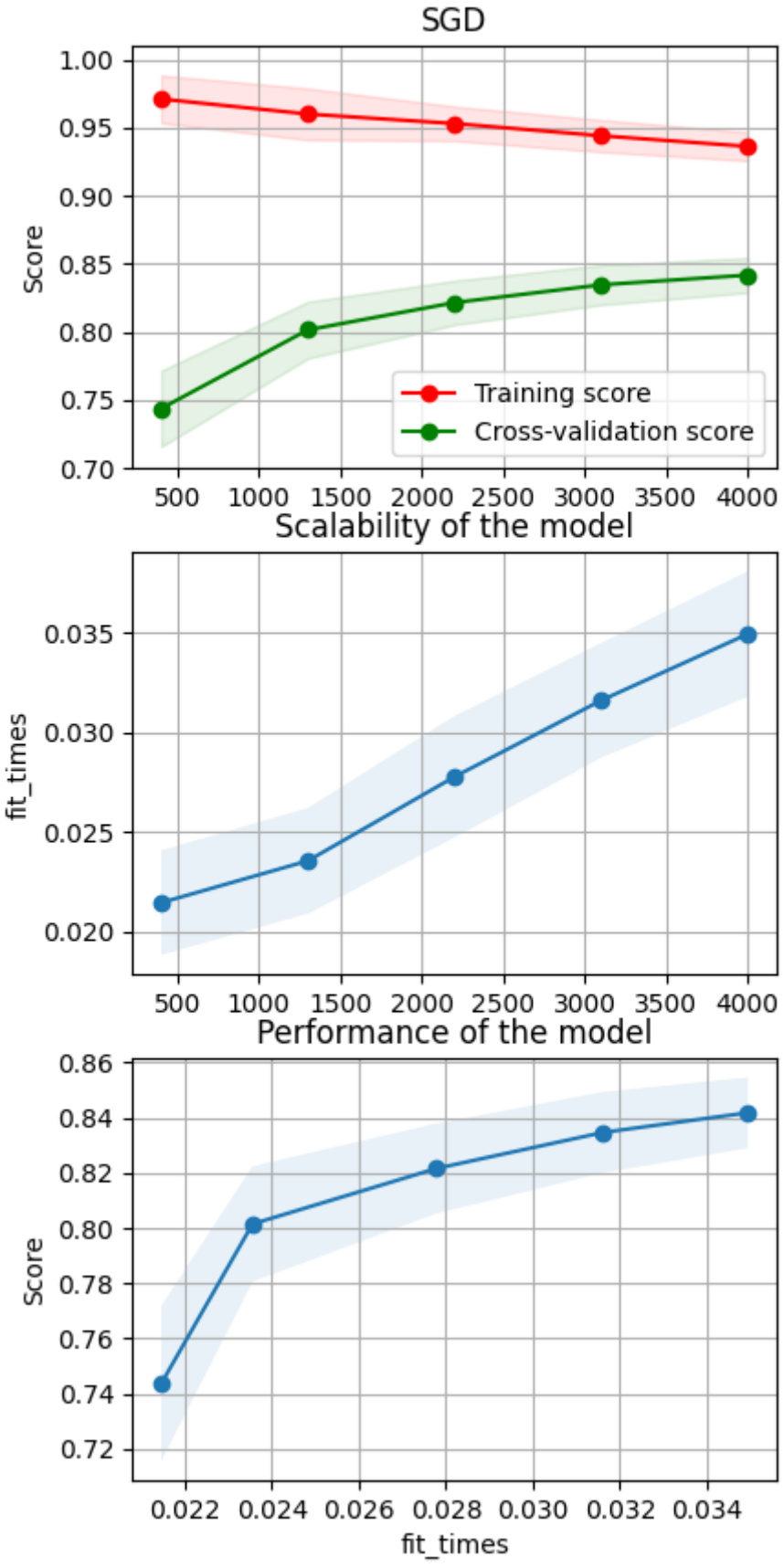
Accuracy of model after training:

```
Accuracy: 87.2%
```



Confusion matrix shows that during testing model classified:

1. 2110 texts as positive
2. 2240 texts as negative
3. 355 texts as positive although they were negative
4. 287 text as negative although they were positive


SGD
Score
Training score
Cross-validation score
Scalability of the model
fit_times
Performance of the model
Score
fit_times

Conclusion

Training data and validation data after some epochs convergence. Also scalability and performance of model increase in time, but the disadvantage models based on gradient descent used in text classification is that when writing ironic text, classify this review as positive. Although for human it is obviously a negative opinion.
It is possible that a major positive part of training data consists of words like happy, like, etc. They become numbers after conversion to weights. When ironic text occurs then the classifier calculates the probability of class based on weights from previous examples, so it decides that this is positive text.