Lectures 8: Class demo

Contents

Imports

```
import os
import sys

sys.path.append(os.path.join(os.path.abspath(".."), (".."), "code"))
from utils import *
import matplotlib.pyplot as plt
import mglearn
import numpy as np
import pandas as pd
from plotting_functions import *
from sklearn.dummy import DummyClassifier
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.model_selection import cross_val_score, cross_validate, train_tes
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
%matplotlib inline
pd.set_option("display.max_colwidth", 200)
DATA_DIR = os.path.join(os.path.abspath(".."), (".."), "data/")
```

Demo: Model interpretation of linear classifiers

- One of the primary advantage of linear classifiers is their ability to interpret models.
- For example, with the sign and magnitude of learned coefficients we could answer questions such as which features are driving the prediction to which direction.

- We'll demonstrate this by training `LogisticRegression` on the famous IMDB movie review dataset. The dataset is a bit large for demonstration purposes. So I am going to put a big portion of it in the test split to speed things up.

```
imdb_df = pd.read_csv(DATA_DIR + "imdb_master.csv", encoding="ISO-8859-1")
imdb_df.head()
```

	review	sentiment
0	One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are right, as this is exactly what happened with me. The first thing that struck me...	positive
1	A wonderful little production. The filming technique is very unassuming- very old-time-BBC fashion and gives a comforting, and sometimes discomforting, sense of realism to the entire p...	positive
2	I thought this was a wonderful way to spend time on a too hot summer weekend, sitting in the air conditioned theater and watching a light-hearted comedy. The plot is simplistic, but the dialogue i...	positive
3	Basically there's a family where a little boy (Jake) thinks there's a zombie in his closet & his parents are fighting all the time. This movie is slower than a soap opera... and suddenl...	negative
4	Petter Mattei's "Love in the Time of Money" is a visually stunning film to watch. Mr. Mattei offers us a vivid portrait about human relations. This is a movie that seems to be telling us what mone...	positive

Let's clean up the data a bit.

```
import re

def replace_tags(doc):
    doc = doc.replace(r"<br />", " ")
    doc = re.sub(r"https://\S*", "", doc)
    return doc
```

```
imdb_df["review_pp"] = imdb_df["review"].apply(replace_tags)
```

Are we breaking the Golden rule here?

Let's split the data and create bag of words representation.

```
train_df, test_df = train_test_split(imdb_df, test_size=0.9, random_state=123)
X_train, y_train = train_df["review_pp"], train_df["sentiment"]
X_test, y_test = test_df["review_pp"], test_df["sentiment"]
train_df.shape
```

```
(5000, 3)
```

```
vec = CountVectorizer(stop_words="english")
bow = vec.fit_transform(X_train)
bow
```

```
<Compressed Sparse Row sparse matrix of dtype 'int64'
	with 439384 stored elements and shape (5000, 38867)>
```

Examining the vocabulary

- The vocabulary (mapping from feature indices to actual words) can be obtained using `get_feature_names_out()` on the `CountVectorizer` object.

```
vocab = vec.get_feature_names_out()
```

```
vocab.shape
```

```
(38867,)
```

```
vocab[0:10]  # first few words
```

```
array(['00', '000', '007', '0079', '0080', '0083', '00pm', '00s', '01',
       '0126'], dtype=object)
```

```
vocab[2000:2010]  # some middle words
```

```
array(['apprehensive', 'apprentice', 'approach', 'approached',
       'approaches', 'approaching', 'appropriate', 'appropriated',
       'appropriately', 'approval'], dtype=object)
```

```
vocab[::500]  # words with a step of 500
```

```
array(['00', 'aaja', 'affection', 'ambrosine', 'apprehensive', 'attract',
       'barbara', 'bereavement', 'blore', 'brazenly', 'businessman',
       'carrel', 'chatterjee', 'claudio', 'commanding', 'consumed',
       'cramped', 'cynic', 'defining', 'deviates', 'displaced',
       'dramatized', 'edie', 'enforced', 'evolving', 'fanatically',
       'fingertips', 'formal', 'gaffers', 'giogio', 'gravitas',
       'halliday', 'heist', 'hoot', 'iliad', 'infiltrate', 'investment',
       'jobson', 'kidnappee', 'landsbury', 'licentious', 'lousiest',
       'malã', 'maã', 'mice', 'molla', 'museum', 'newtonian',
       'obsessiveness', 'outbursts', 'parapsychologist', 'perpetuates',
       'plasters', 'powers', 'property', 'rabies', 'reclined', 'renters',
       'ridiculous', 'rube', 'sayid', 'select', 'shivers', 'skinheads',
       'sohail', 'spot', 'stomaches', 'suitcase', 'syrupy', 'terrorist',
       'tolerance', 'triangular', 'unbidden', 'unrevealed', 'verneuil',
       'walrus', 'wilcox', 'xxxxviii'], dtype=object)
```

```
y_train.value_counts()
```

```
sentiment
positive    2517
negative    2483
Name: count, dtype: int64
```

Model building on the dataset

First let's try `DummyClassifier` on the dataset.

```
dummy = DummyClassifier()
scores = cross_validate(dummy, X_train, y_train, return_train_score=True)
pd.DataFrame(scores)
```

	fit_time	score_time	test_score	train_score
0	0.001099	0.000938	0.504	0.50325
1	0.000945	0.000799	0.504	0.50325
2	0.000909	0.000825	0.503	0.50350
3	0.000986	0.000831	0.503	0.50350
4	0.000915	0.000827	0.503	0.50350

We have a balanced dataset. So the `DummyClassifier` score is around 0.5.

Now let's try logistic regression.

```
pipe_lr = make_pipeline(
    CountVectorizer(stop_words="english"),
    LogisticRegression(max_iter=1000),
)
scores = cross_validate(pipe_lr, X_train, y_train, return_train_score=True)
pd.DataFrame(scores)
```

	fit_time	score_time	test_score	train_score
0	0.385554	0.059766	0.828	0.99975
1	0.377250	0.061282	0.830	0.99975
2	0.383834	0.059718	0.848	0.99975
3	0.370757	0.058832	0.833	1.00000
4	0.372611	0.062104	0.840	0.99975

Seems like we are overfitting. Let's optimize the hyperparameter `C` of LR and `max_features` of `CountVectorizer`.

```
from scipy.stats import loguniform, randint, uniform
from sklearn.model_selection import RandomizedSearchCV

param_dist = {
    "countvectorizer__max_features": randint(10, len(vocab)),
    "logisticregression__C": loguniform(1e-3, 1e3)
}
pipe_lr = make_pipeline(CountVectorizer(stop_words="english"), LogisticRegress
random_search = RandomizedSearchCV(pipe_lr, param_dist, n_iter=10, n_jobs=-1,
random_search.fit(X_train, y_train)
```

```
▸ RandomizedSearchCV
▸ best_estimator_: Pipeline
    ▸ CountVectorizer
    ▸ LogisticRegression
```

```
pd.DataFrame(random_search.cv_results_)
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_count
0	0.588244	0.021243	0.110041	0.006330	
1	0.657085	0.028550	0.110500	0.006087	
2	0.501113	0.022524	0.105766	0.008775	
3	0.519245	0.042000	0.126679	0.018991	
4	0.503987	0.069851	0.122257	0.018959	
5	0.552576	0.008925	0.106016	0.012166	
6	0.732404	0.057531	0.130433	0.014476	
7	0.743628	0.077521	0.100707	0.004814	
8	0.572081	0.014627	0.104338	0.004348	
9	0.468955	0.052592	0.088705	0.014833	

10 rows × 22 columns

```
cv_scores = random_search.cv_results_['mean_test_score']
train_scores = random_search.cv_results_['mean_train_score']
countvec_max_features = random_search.cv_results_['param_countvectorizer__max_
```

```
pd.DataFrame(random_search.cv_results_)[
    [
        "mean_test_score",
        "mean_train_score",
        "param_logisticregression__C",
        "param_countvectorizer__max_features",
        "mean_fit_time",
        "rank_test_score",
    ]
].set_index("rank_test_score").sort_index()
```

	mean_test_score	mean_train_score	param_logisticregression__
rank_test_score			
1	0.8400	0.99700	0.17833
2	0.8396	0.99925	0.31768
3	0.8386	0.96020	0.02784
4	0.8346	1.00000	356.0177
5	0.8318	1.00000	5.46987
6	0.8246	1.00000	6.81921
7	0.8238	0.88735	0.00470
8	0.8236	1.00000	6.84976
9	0.8178	0.87315	0.00292
10	0.8004	1.00000	12.79360

Let’s train a model on the full training set with the optimized hyperparameter values.

```
best_model = random_search.best_estimator_
```

```
best_model
```

```
▸ Pipeline
  ▸ CountVectorizer
  ▸ LogisticRegression
```

Examining learned coefficients

- The learned coefficients are exposed by the `coef_` attribute of [LogisticRegression](#) object.

```python
# Get feature names
feature_names = best_model.named_steps['countvectorizer'].get_feature_names_ou

# Get coefficients
coeffs = best_model.named_steps["logisticregression"].coef_.flatten()
```

```python
word_coeff_df = pd.DataFrame(coeffs, index=feature_names, columns=["Coefficien
word_coeff_df
```

	Coefficient
00	0.067353
000	0.105312
007	0.006590
0083	0.027134
00s	-0.052512
...	...
zwick	0.019599
zyada	-0.026069
zzzzip	-0.000101
zzzzz	-0.031992
â½	-0.010470

30575 rows × 1 columns

- Let's sort the coefficients in descending order.
- Interpretation
  - if $w_j > 0$ then increasing x_{ij} moves us toward predicting $+1$.
  - if $w_j < 0$ then increasing x_{ij} moves us toward predicting -1.

```
word_coeff_df.sort_values(by="Coefficient", ascending=False)
```

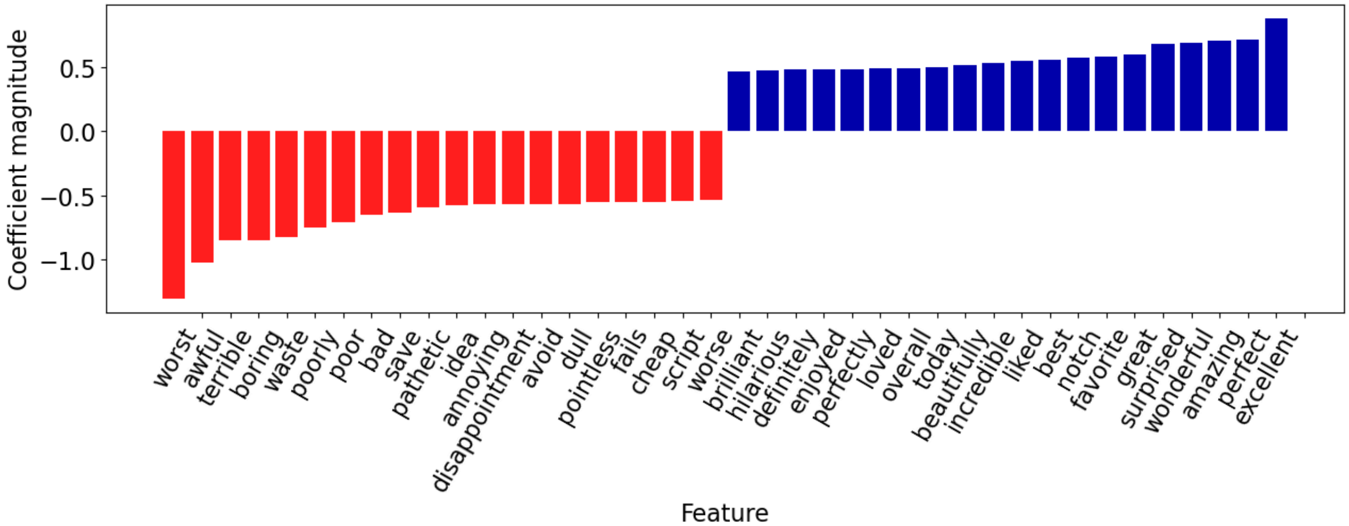
	Coefficient
excellent	0.876824
perfect	0.713843
amazing	0.707115
wonderful	0.685750
surprised	0.679117
...	...
waste	-0.823069
boring	-0.847032
terrible	-0.847147
awful	-1.026551
worst	-1.303253

30575 rows × 1 columns

- The coefficients make sense!

Let's visualize the top 20 features.

```
mglearn.tools.visualize_coefficients(coeffs, feature_names, n_top_features=20)
```



Let's explore prediction of the following new review.

```
fake_reviews = ["It got a bit boring at times but the direction was excellent
 "The plot was shallower than a kiddie pool in a drought, but hey, at least we
]
```

Let's get prediction probability scores of the fake review.

```
best_model.predict(fake_reviews)
```

```
array(['positive', 'negative'], dtype=object)
```

```
# Get prediction probabilities for fake reviews
best_model.predict_proba(fake_reviews)
```

```
array([[0.13615126, 0.86384874],
       [0.72517628, 0.27482372]])
```

```
best_model.classes_
```

```
array(['negative', 'positive'], dtype=object)
```

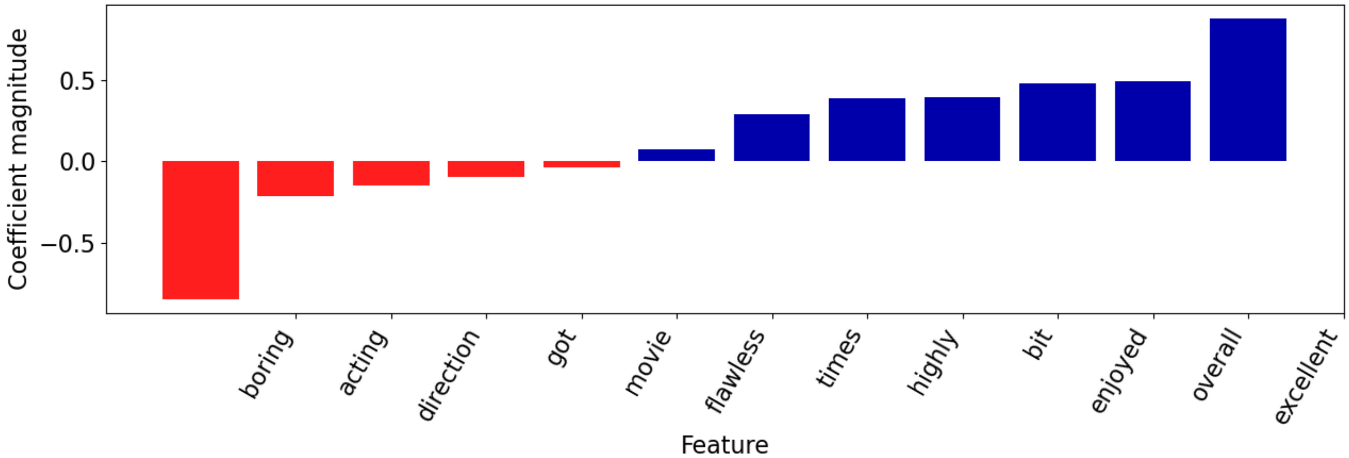
We can find which of the vocabulary words are present in this review:

```python
def plot_coeff_example(model, review, coeffs, feature_names, n_top_feats=6):
    print(review)
    feat_vec = model.named_steps["countvectorizer"].transform([review])
    words_in_ex = feat_vec.toarray().ravel().astype(bool)

    ex_df = pd.DataFrame(
        data=coeffs[words_in_ex],
        index=np.array(feature_names)[words_in_ex],
        columns=["Coefficient"],
    )
    mglearn.tools.visualize_coefficients(
    coeffs[words_in_ex], np.array(feature_names)[words_in_ex], n_top_features=
    )
    return ex_df.sort_values(by=["Coefficient"], ascending=False)
```

```python
plot_coeff_example(best_model, fake_reviews[0], coeffs, feature_names)
```

```
It got a bit boring at times but the direction was excellent and the acting wa
```

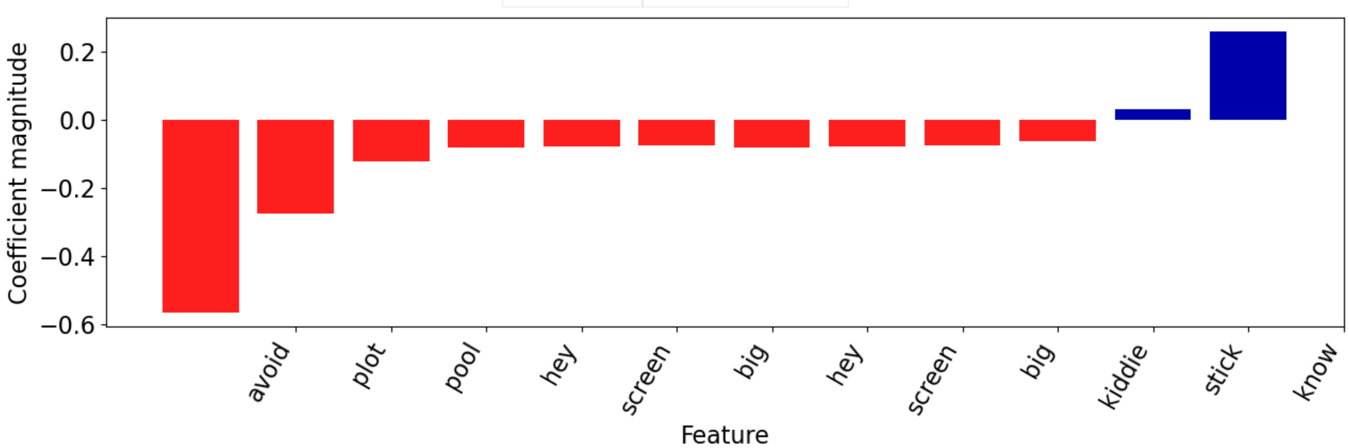
	Coefficient
excellent	0.876824
overall	0.492946
enjoyed	0.481338
bit	0.397143
highly	0.386792
times	0.287180
recommend	0.182552
flawless	0.071724
movie	-0.035484
got	-0.097060
direction	-0.146983
acting	-0.213584
boring	-0.847032




```
plot_coeff_example(best_model, fake_reviews[1], coeffs, feature_names)
```

```
The plot was shallower than a kiddie pool in a drought, but hey, at least we nc
```

	Coefficient
know	0.256913
stick	0.029108
kiddie	-0.062415
big	-0.077145
screen	-0.078166
hey	-0.083086
pool	-0.123016
plot	-0.276647
avoid	-0.567104



Most positive review

- Remember that you can look at the probabilities (confidence) of the classifier's prediction using the `model.predict_proba` method.
- Can we find the reviews where our classifier is most certain or least certain?

```
# only get probabilities associated with pos class
pos_probs = best_model.predict_proba(X_train)[
    :, 1
]  # only get probabilities associated with pos class
pos_probs
```

```
array([0.98488155, 0.17244179, 0.96027595, ..., 0.80965294, 0.91813092,
       0.00243243])
```

What's the index of the example where the classifier is most certain (highest `predict_proba` score for positive)?

```
most_positive_id = np.argmax(pos_probs)
```

```
print("True target: %s\n" % (y_train.iloc[most_positive_id]))
print("Predicted target: %s\n" % (best_model.predict(X_train.iloc[[most_positi
print("Prediction probability: %0.4f" % (pos_probs[most_positive_id]))
```

```
True target: positive

Predicted target: positive

Prediction probability: 1.0000
```

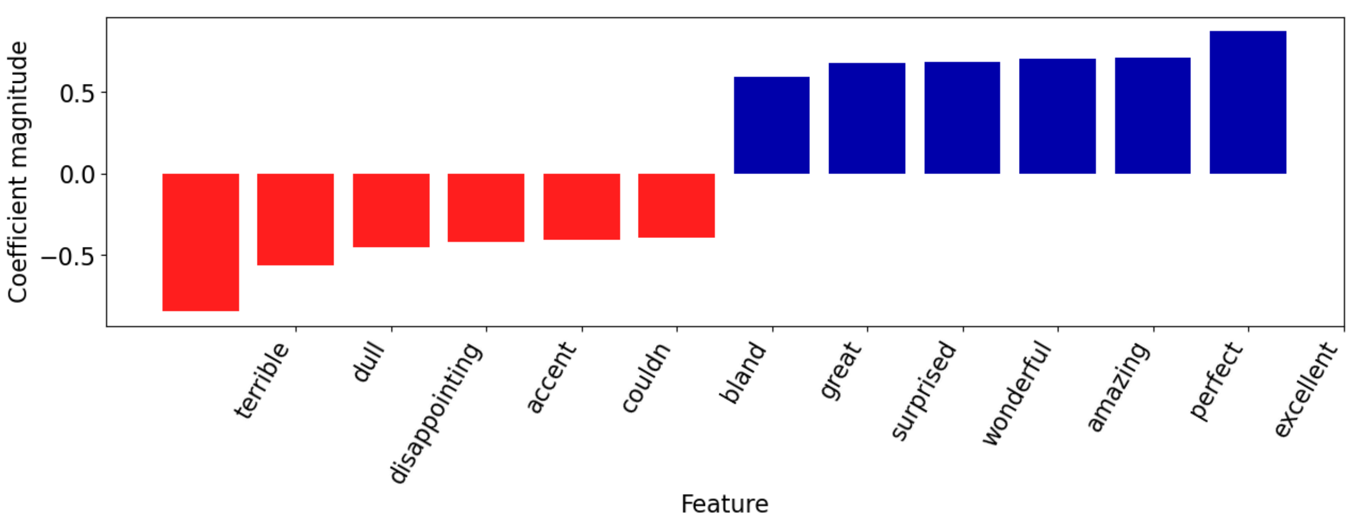
Let's examine the features associated with the review.

```
plot_coeff_example(best_model, X_train.iloc[most_positive_id], coeffs, feature
```

```
This is an awesome Amicus horror anthology, with 3 great stories, and fantasti
```

	Coefficient
excellent	0.876824
perfect	0.713843
amazing	0.707115
wonderful	0.685750
surprised	0.679117
...	...
couldn	-0.409726
accent	-0.419552
disappointing	-0.451959
dull	-0.565241
terrible	-0.847147

173 rows × 1 columns



The review has both positive and negative words but the words with **positive** coefficients win in this case!

Most negative review

```
neg_probs = best_model.predict_proba(X_train)[
    :, 0
]  # only get probabilities associated with neg class
neg_probs
```

```
array([0.01511845, 0.82755821, 0.03972405, ..., 0.19034706, 0.08186908,
       0.99756757])
```

```
most_negative_id = np.argmax(neg_probs)
```

```
print("Review: %s\n" % (X_train.iloc[[most_negative_id]]))
print("True target: %s\n" % (y_train.iloc[most_negative_id]))
print("Predicted target: %s\n" % (best_model.predict(X_train.iloc[[most_negati
print("Prediction probability: %0.4f" % (neg_probs[most_negative_id]))
```

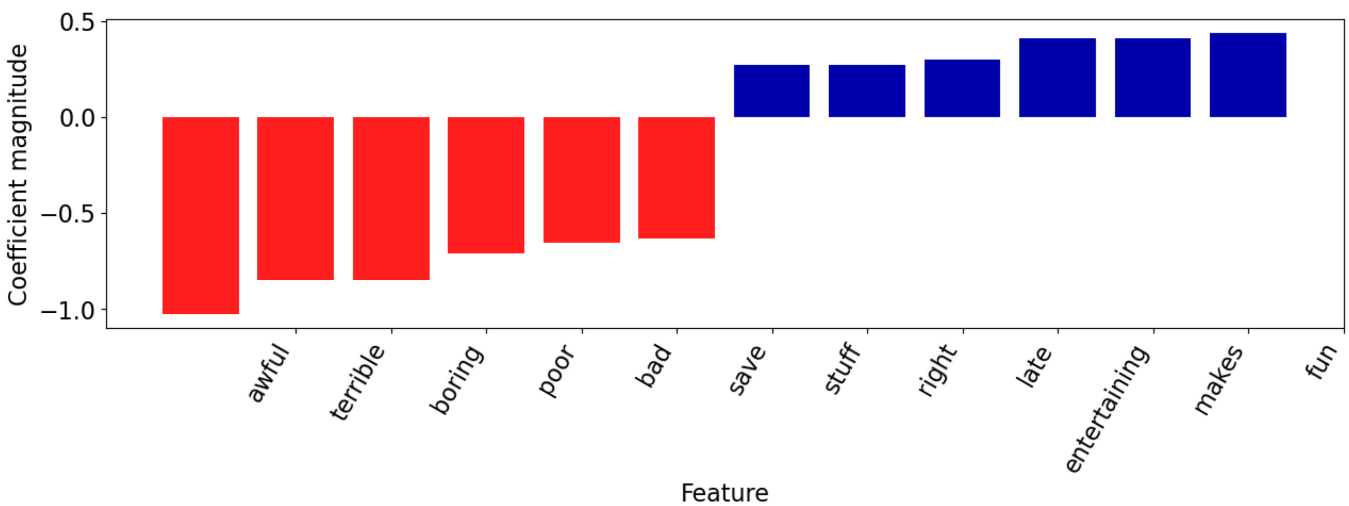
```
Review: 13452    Zombi 3 starts as a group of heavily armed men steal a experin
Name: review_pp, dtype: object

True target: negative

Predicted target: negative

Prediction probability: 1.0000
```

```
plot_coeff_example(best_model, X_train.iloc[most_negative_id], coeffs, feature
```

```
Zombi 3 starts as a group of heavily armed men steal a experimental chemical de
```

	Coefficient
fun	0.434700
makes	0.407873
entertaining	0.407167
late	0.299810
right	0.272165
...	...
bad	-0.652233
poor	-0.712592
boring	-0.847032
terrible	-0.847147
awful	-1.026551

317 rows × 1 columns



The review has both positive and negative words but the words with negative coefficients win in this case!

❓❓ Questions for you

Question for you to ponder on

- Is it possible to identify most important features using k-NNs? What about decision trees?