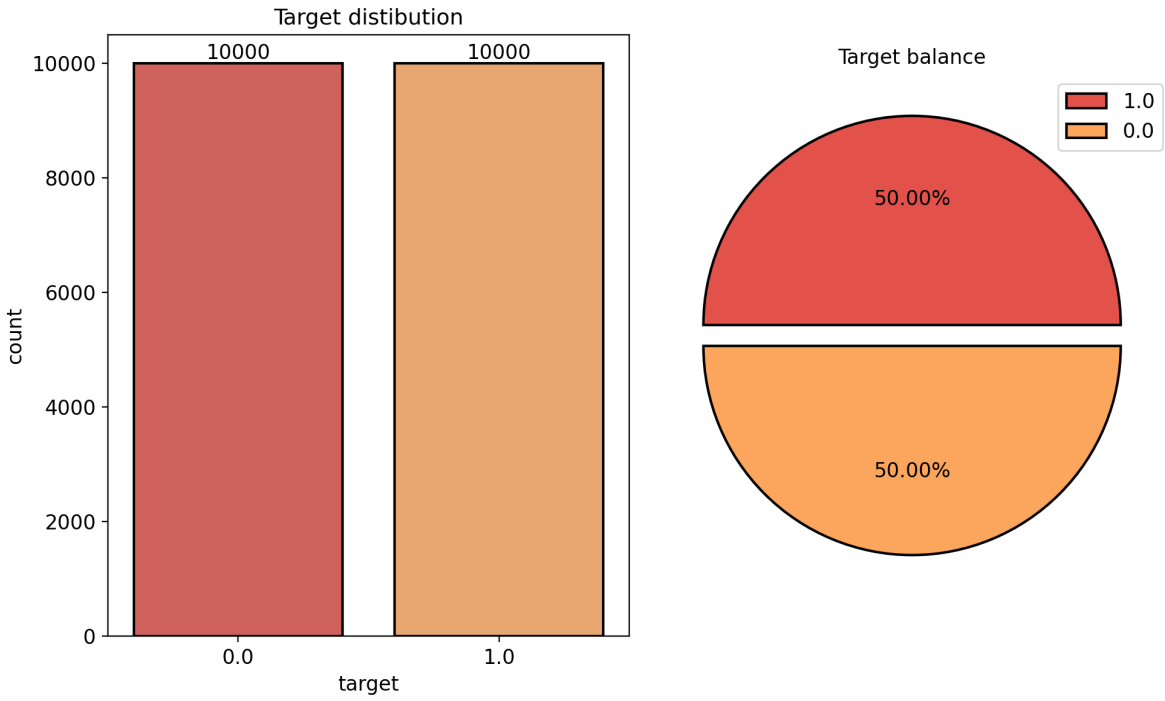
Отчёт

Фактически это просто выжимка из ноутбуков, я опишу вкратце, что я делал, зачем, и чего достиг, за подробностями отсылаю к тетрадкам в репозитории или на кеггле. А ещё при переводе в pdf ломаются некоторые выводы, поэтому я заменил их картинками в некоторых случаях

Данные

С самими данными я не делал ничего. Для русского я этого не люблю делать вообще никогда, для английского я допустил, что это может улучшить качество, проделал, не улучшило, всё записано в метриках ниже. К тому же почти вся предобработка, что мне нужна уже возможна, благодаря `TfidfVectorizer`
Данные сами по себе довольно чистые, датасет полностью сбалансирован, это очень приятно.

In [189…
```
check_imbalance(df_train, "target")
```



Исходя из этого, за оптимизируемую метрику я брал accuracy, а также немного смотрел на ROC-AUC. Если бы был дисбаланс, пришлось бы подумать ещё

Модели

Я пробовал и классические модели, и нейросети, можно почитать в соответствующих ноутбуках. В любом случае меня интересовали модели, которые
1) работают достаточно быстро, но это на самом деле почти все, если текст всего один
2) сравнительно быстро обучаются, потому что в любом случае будет оптимизация, не хочется сидеть над ней вечно 3) предсказывают вероятности, это фиксится калибровкой, конечно, но она вредит пунктам (1) и (2) 4) мало весит, по этому

критерию отлетают почти все нейросети, что грустно 5) хорошо интерпретируются, но это худо-бедно можно сделать для всех моделей, у меня была задумка делать это только для входного текста, а не анализ вообще всех признаков

По всем этим критериям идеально подходит логистическая регрессия, хотя я пробовал и прочие методы, конечно, примерные метрики внизу

In [ ]:
```
metrics_df = pd.DataFrame(metrics)
metrics_df[metrics_df.dataset == "val"].sort_values("accuracy", ascending=False)
```

[24]:

	model	dataset	inference_time	accuracy	precision	recall	f1	roc_auc	auc_pr
16	bigram_svm	val	1.950951	0.8938	0.8938	0.893896	0.893794	0.000000	0.000000
6	unigram_proba_svc	val	49.397370	0.8884	0.8884	0.888481	0.888394	0.956223	0.954352
4	unigram_svm	val	0.841167	0.8874	0.8874	0.887494	0.887393	0.000000	0.000000
2	unigram_logreg	val	0.920446	0.8862	0.8862	0.886252	0.886196	0.952450	0.951335
14	bigram_logreg	val	1.960605	0.8828	0.8828	0.882825	0.882798	0.948548	0.948153
10	unigram_catboost	val	0.972381	0.8620	0.8620	0.862317	0.861970	0.938907	0.938059
12	bigram_nb	val	2.144800	0.8562	0.8562	0.864579	0.855369	0.945955	0.941801
8	unigram_random_forest	val	1.283854	0.8454	0.8454	0.845429	0.845397	0.923196	0.915395
18	bigram_random_forest	val	3.625016	0.8450	0.8450	0.845458	0.844949	0.922659	0.914399
0	unigram_nb	val	0.849646	0.8404	0.8404	0.843894	0.839994	0.926143	0.914936

Всё это было сделано при помощи Tfidf, w2v я не уважаю, и в предобученном виде, и в виде fasttext'a, но его я тоже попробовал, через усреднение всех векторов слов, вышло хуже

In [ ]:
```
metrics_df = pd.DataFrame(metrics)
metrics_df[metrics_df.dataset == "val"].sort_values("accuracy", ascending=False)
```

[254]:

	model	dataset	inference_time	accuracy	precision	recall	f1	roc_auc	auc_pr
4	w2v_svm	val	0.007047	0.8230	0.8230	0.823023	0.822997	0.000000	0.000000
2	w2v_logreg	val	0.009725	0.8172	0.8172	0.817201	0.817200	0.895014	0.890229
8	w2v_catboost	val	0.073740	0.8024	0.8024	0.802485	0.802386	0.886508	0.882440
6	w2v_random_forest	val	0.147221	0.7752	0.7752	0.775285	0.775183	0.855443	0.849151
0	w2v_nb	val	0.060080	0.6926	0.6926	0.694173	0.691976	0.771788	0.743388

Из нейросетей я пробовал LSTM, и ещё некоторые архитектуры, сразу же их дообучил, внизу статистика с самым главным. Жирным модели, которые я планировал использовать на сайте, но использовал только логрег:

Models | Model | Accuracy | ROC-AUC | |----------|---|---| | **XLNet** | **93.6** | 0.981 | | RoBERTa | 93.8 | - | | BERT | 92.4 | - | | LSTM | ~86 | - | | **Optimized LogReg** | **90.5** | 0.966 | | LogReg | 88.3 | 0.952 | | SVM | 89.4 | - | | CatBoost | 88.7 | 0.955 | | BernoulliNB | 85.6 | 0.946 | | RandomForest | 84.5 | 0.923 |

Не все из них замерялись на тесте, особенно касается нейросетей, но данные действительно отлично сбалансированы, метрики почти совпадают. Впрочем, даже

базовых моделей хватает, чтобы побить скор, указанный в статье, которую прилагали к заданию

Тюнинг

Для нейросеток я пытался подбирать число выходных слоёв и коэффициент дропаута, но положительного эффекта не было. Помог разве что автоматический подбор лёрнинг рейта. В любом случае мне кажется, что их иожно дообучить ещё лучше

Для логрега я использовал оптуну на кросс-валидации со всеми мыслимыми параметрами, в том числе и для Tfidf, ниже лучшие параметры

In [33]:
```
logreg_study.best_value, logreg_study.best_params
```

Out[33]:
```
(0.90672,
 {'penalty': 'l2',
  'C': 13.991882149161302,
  'tol': 2.3655997265491263e-06,
  'fit_intercept': True,
  'max_iter': 32,
  'l2_solver': 'saga',
  'use_idf': True,
  'analyzer': 'word',
  'stop_words': None,
  'lowercase': True,
  'min_df': 5,
  'token_pattern': '\\S+',
  'ngram_range': 2})
```

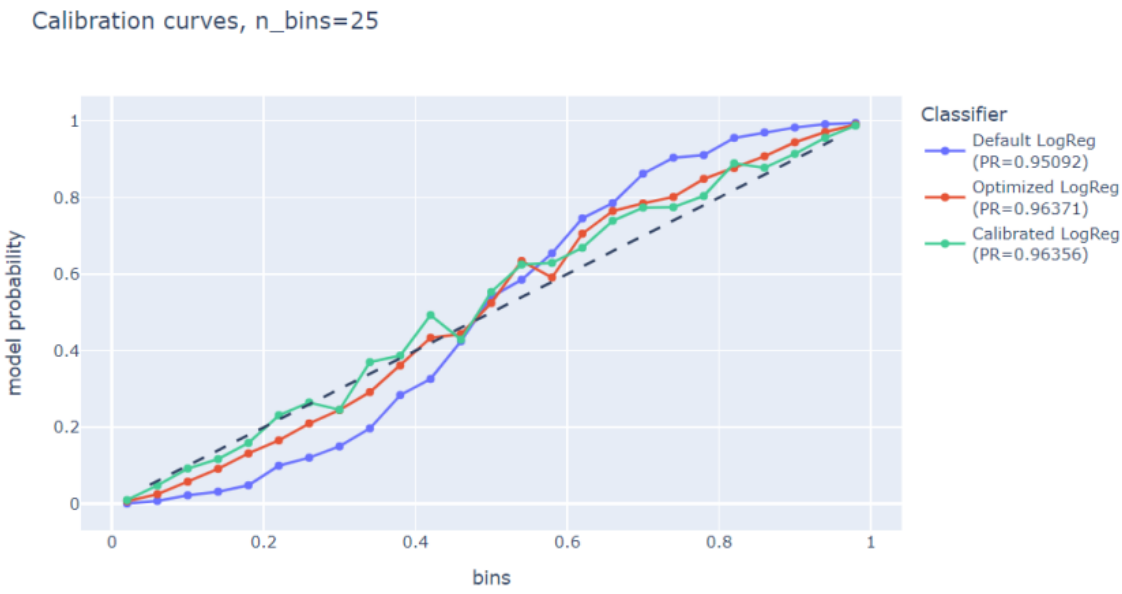
Пороги я тоже отбирал, но смысла в этом мало, выборка, опять же, сбалансирована. Ниже сравнение регрессии с дефолтными параметрами и оптимизированной

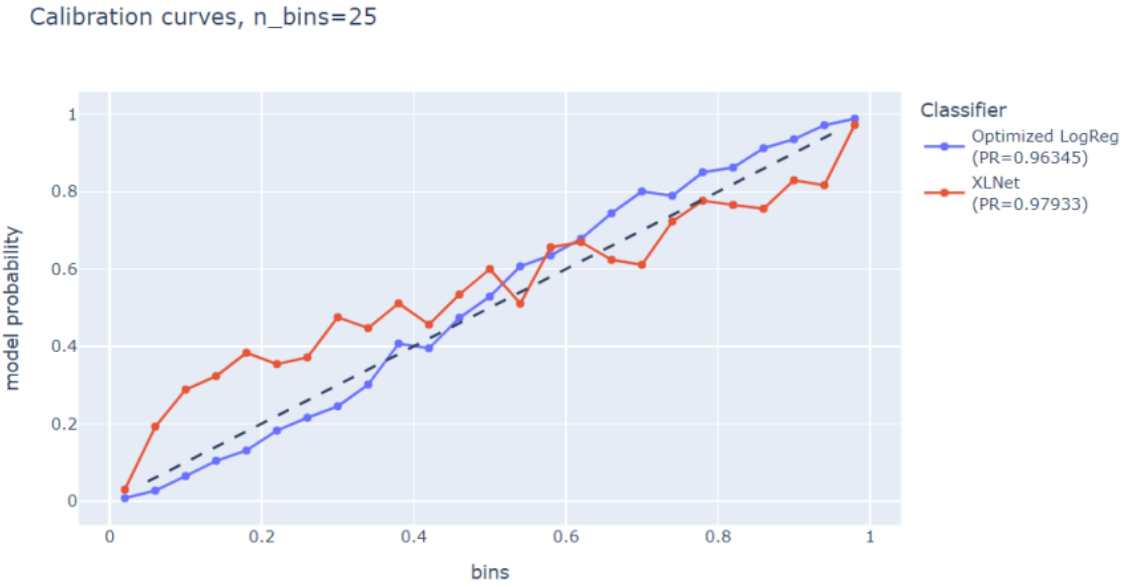
In [20]:
```
print("Default")
print(f"Accuracy: {accuracy_score(proba > 0.5, y_test)}")
print(f"ROC-AUC: {roc_auc_score(y_test, proba)}\n")

print("Optimized")
print(f"Accuracy: {accuracy_score(optimized_proba > 0.5, y_test)}")
print("Optimized with chosen threshold")
print(f"Accuracy: {accuracy_score(optimized_proba > np.mean(best_t), y_test)}")
print(f"ROC-AUC: {roc_auc_score(y_test, optimized_proba)}")
```

```
Default
Accuracy: 0.883
ROC-AUC: 0.9523968319999999

Optimized
Accuracy: 0.90488
Optimized with chosen threshold
Accuracy: 0.90512
ROC-AUC: 0.9655946304
```

И конечно, поскольку нам нужны вероятности, на это я тоже посмотрел на калибровочных кривых

In [ ]:
```
plot_calibration_curve(y_test, {"Default LogReg": proba,
                                "Optimized LogReg": optimized_proba,
                                "Calibrated LogReg": calibrated_proba},
                       n_bins=25)
```





Как видно, оптимизация очень даже имела смысл. Выиграли и по аккураси, и по вероятностям. Логрег в этом отношении даже лучше, чем XLNet, судя по графику

Интерпретация модели

Это я сделал только для логистической регрессии, потому что к этому моменту уже отчался использовать нейросети. Там их достаточно легко достать через веса самой регрессии. У меня была мысль вместе с предсказанием показывать, как модель к нему пришла. Я встроил это на самом сайте, лучше там смотреть, там и график интерактивный, хотя мне кажется, что из-за него всё так лагает

В ноутбуках интерфейс тоже есть, можно на него посмотреть

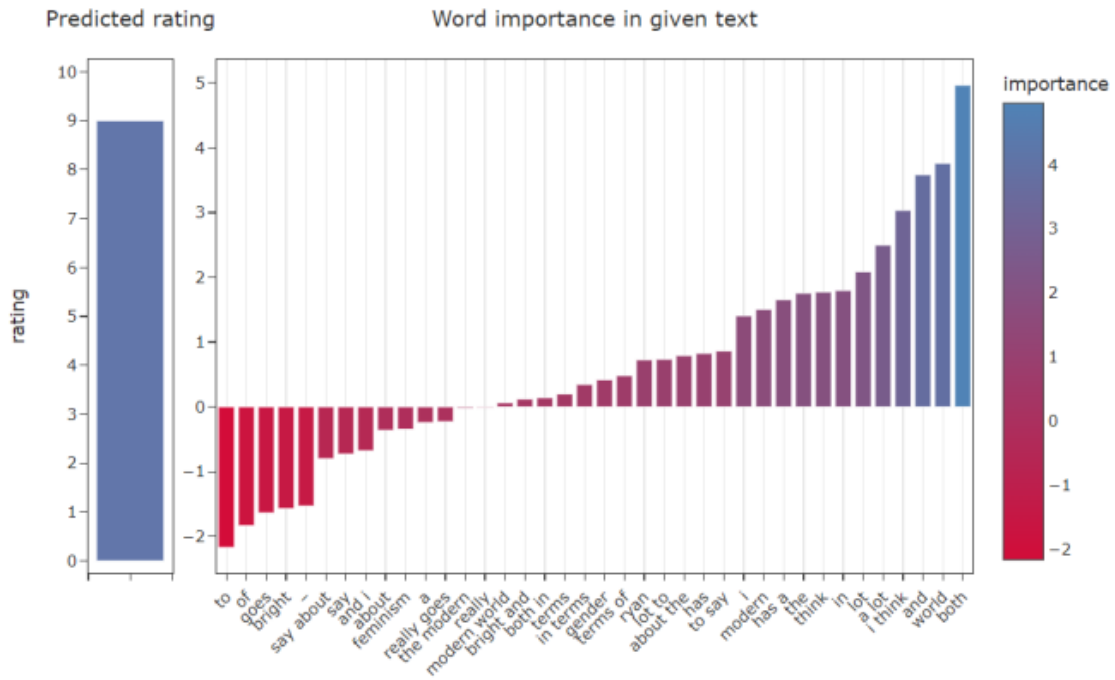
In [39]:
```
%%time
print(rating_estimate("This movie is utter dogshit, how could anyone possibly like
print(rating_estimate("I enjoyed it a lot however"))
print(rating_estimate("It was mediocre for the most part, but the ending was brill
```

```
(1, 'negative')
(10, 'positive')
(4, 'negative')
CPU times: user 8.5 ms, sys: 1.94 ms, total: 10.4 ms
Wall time: 9.18 ms
```

Ну и посмотрим на какой-нибудь относительно сложный текст

In [ ]:
```
text = "It’s funny, it’s bright and uplifting, and I think has a lot to say about t

print(ml_infer(text))
print(rating_estimate(text))
compute_message_feature_importance(text, rating_estimate(text)[0])
```

```
(9, 'positive')
```



Получается вроде бы довольно неплохо, ниже оригинальный отзыв

В целом это всё, но в репозитории есть ещё полезные ссылки, а в двух тетрадках подробно описан почти каждый шаг