Amazon Product Evaluation Final Predictions

Dataset used: Toys and Games

Due Date: May 14

Joris Chomarat
Halil Arici
Gyaan Antia

Recap: Features used

	asin	###
1	weighted avg. compound text	$\in \mathbb{Q}$
2	weighted avg. compound summary	$\in \mathbb{Q}$
3	std. dev. text	$\in \mathbb{Q}$
4	std. dev. summary	$\in \mathbb{Q}$
5	verification percentage	$[0 \dots 1] \in \mathbb{Q}$
6	amount of reviews	$\in \mathbb{N}^+$
7	avg. star rating	$[1 \dots 5] \in \mathbb{N}$
	awesomeness	$[0,1]$

1) 2) 3) 4) on next slide

5) Verification percentage

Self-explanatory. Percentage of verified reviews out of all reviews of one product.

6) Amount of reviews

Self-explanatory. Amount of reviews a product has.

7) Average star rating

Summary data in dataset contains star rating left by reviewer, if no summary was left. The average of all ratings from these reviews was taken. If no review of a product has a star rating, the lowest possible Amazon star rating of ”1” is assigned.

Recap: Features used (continued)

1) *Weighted average compound text*

Compound score of NLP, weighted according to whether review is verified, has image, number of votes and age. Then the average is taken over all weighted review scores of a product.

$$comp_{w,avg} = \frac{1}{n}\sum_{i=1}^{n} comp_i * w_{ver,i} * w_{img,i} * w_{vote,i} * w_{age,i}$$

2) *Weighted average compound summary*

Same procedure as with 1), but star ratings excluded. If only star ratings are available for a product, “0” is assigned.

3) *Standard deviation compound text*

Standard deviation of all unweighted compound scores of all reviews of a product.

4) *Standard deviation compound summary*

Same procedure as with 3), but star ratings excluded. If only star ratings are available for a product, “0” is assigned.

Compound score

The compound score is given by using the **Vader Sentiment Analyzer** for NLP. It is calculated by the summation of the valence scores of each word in the lexicon. Then it is normalized: -1 is the most negative and +1 is the most positive score. It can be called a normalized, weighted composite score.

Weights

Verification: 1 if not verified, 1.5 if verified

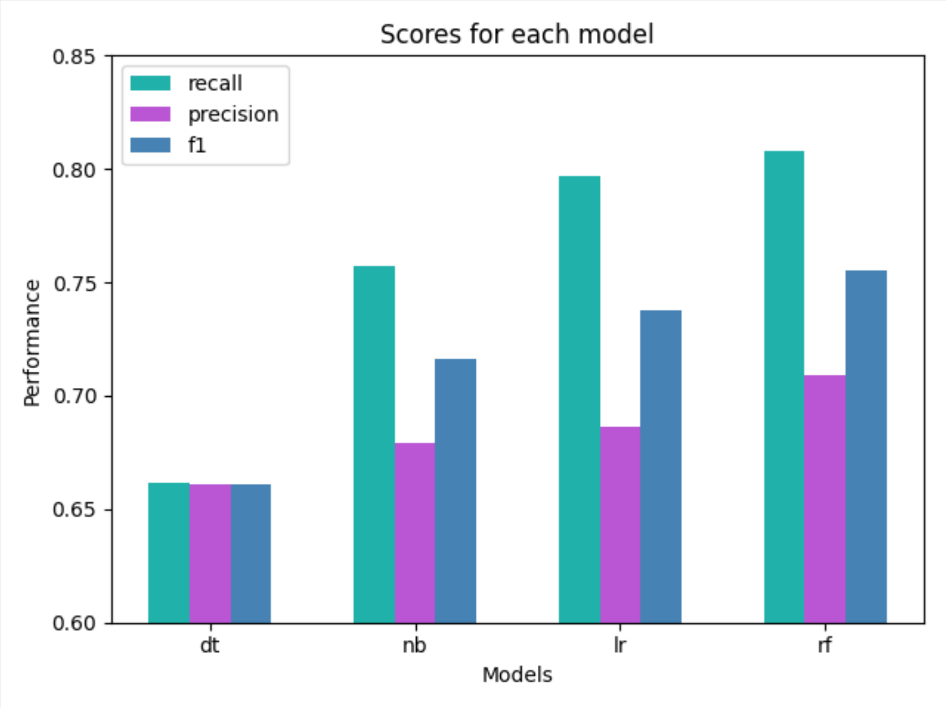
Image: 1 if no image, 1.3 if one image or more

Votes: $w_{vote} = 1 + \frac{\ln(curr+1)}{\ln(\max + 1.1)}$, *curr* being the votes on the current review, *max* being the highest number of votes on any review of this product

Age: $w_{age} = \frac{0.01}{s\ per\ year}(curr - first) + 1$, *curr* being the age (in seconds) of the current review, *first* being the age of the first review on this product

Recap: Algorithms & Results

- Model performance was evaluated using 10-fold cross validation for each model
- Classifier hyper-parameters were optimized using a grid search algorithm maximizing the F1 score
  - Hyper-parameters not mentioned on next slides were kept at their default value
  - Where possible, the amount of CPU cores was increased to the maximum
- Classifiers evaluated:
  - Decision Tree (*model_dt*)
  - Naïve Bayes (*model_nb*)
  - Logistic Regression (*model_lr*)
  - Random Forest Classifier (*model_rf*)



- based on performance of the optimized classifiers tried out, predictions on *test1* were done with *model_rf*
- Feedback *test1*: F1 score of 0.7176 achieved

Recap: possible next steps (as of April 28)

- Classification
  - Implement late fusion to combine multiple classifiers
  - Try out further classifiers, e.g., SVM or KNN
  - Randomize hyper-parameter grid search to find even better performances with more parameters: execute a random search first, a fine grid search in the area of best performance can be performed afterwards
  - Goal: achieve F1 scores > 0.8
- Features
  - Optimize weights used for the weighted average compound scores (iteratively by hand during remaining group part, or as neural network in personal project part)
  - Look into using tentative feature presented in feature deliverable: Product age (time since first review)
  - Look into using other features: Kurtosis and new ideas

Feature vectors

- Changes to feature vectors:
  - *training* and *test* features scaled to a range from -1 to 1, otherwise svm would not converge in our case

  → *model_dt* f1 score decreased by $0.5\ \%$, but wasn't used for predictions either way

  → Outlook: scaling feature vectors increased model performance of a knn classifier covered in the next slides

Classifiers

- The following classifiers were implemented (doing a grid search over them and 10-fold cross validation)

KNN – K Nearest Neighbors

Neighbors	$[5, 10, 15, 20]$	15
Weights	uniform, distance	uniform
Algorithm	auto, ball_tree, kd_tree, brute	brute
Leaf Size	$[10, 20, 30, 40]$	10

SVM – Support Vector Machine

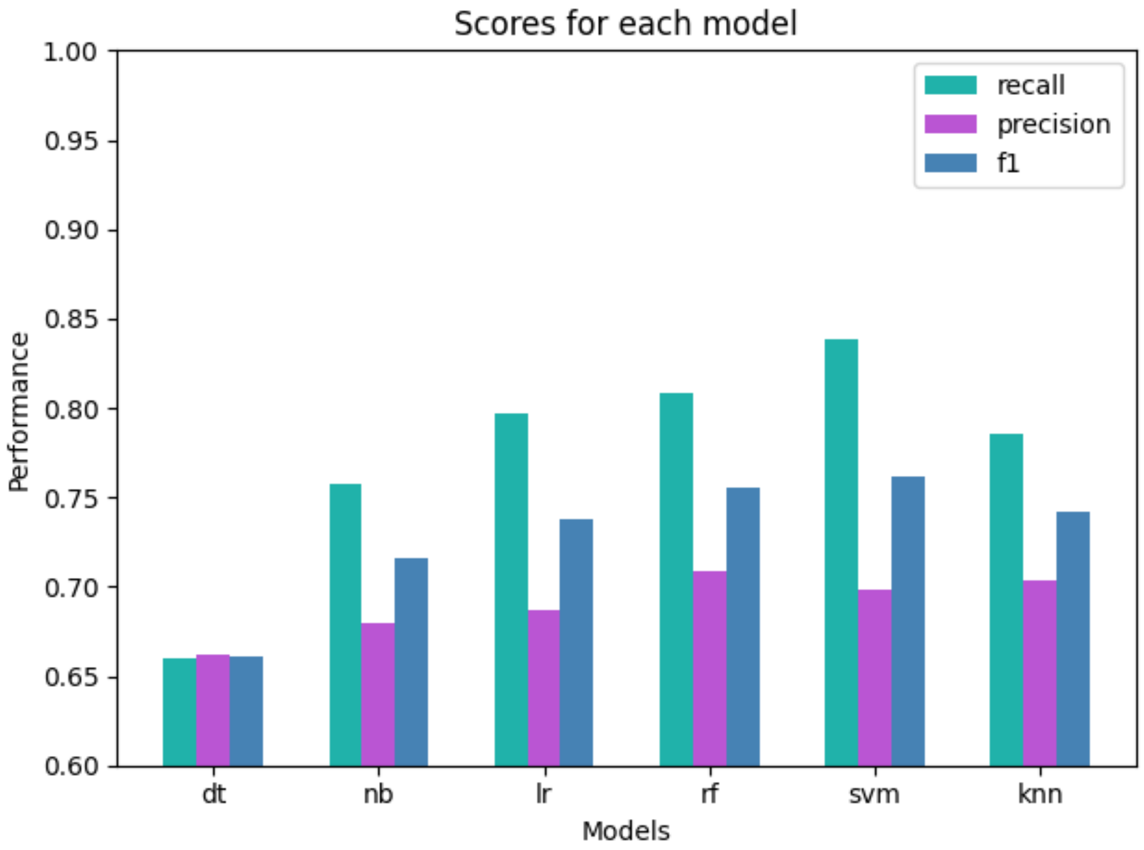
C	$[5, 10, 20]$	20
Gamma	scale, auto	scale
Max_depth	$[0.1, 10^{-3}, 10^{-5}]$	10^{-5}

- Kernel: rbf (other kernels wouldn't converge)
- max_iterations were increased to 1,000,000 so that (nearly) every iteration converged

- Results on next slide

Results (single classifiers)

	Recall	Precision	F1
model_dt	0.6613	0.661	0.6612
model_nb	0.7575	0.6795	0.7162
model_lr	0.7966	0.6866	0.7375
model_rf	0.8081	0.7093	0.7555
model_svm	0.8081	0.6978	0.7616
model_knn	0.7857	0.7033	0.7422



Fusion & Boosting

Late Fusion *(model_vc)*

- models used: *model_lr, model_rf, model_knn*
- svm could not be used, as probabilities are not computed and forcing the model to do so increased computational time too much
- Grid search over model combination:
    - Coarse (steps of 5% weight) first
    - Fine (steps of 1% weight) in vicinity of coarse result
    - Optimal weights found:

LR	RF	KNN
0.12	0.72	0.16

AdaBoost *(model_ab)*

- AdaBoost yielded better results than gradient boost →AdaBoost was used
- Scaling of feature vectors did not have any impact on performance
- Logistic Regression and KNN did not work with AdaBoost → Random Forest used
- Grid searches over ranges getting finer:

estimator__max_depth	[5, 7, 8, 10]	5
n_estimators	[5, 10, 13, 15, 17, 20]	13
learning_rate	[0.6, 0.8, 1, 1.6, 2, 2.5]	2

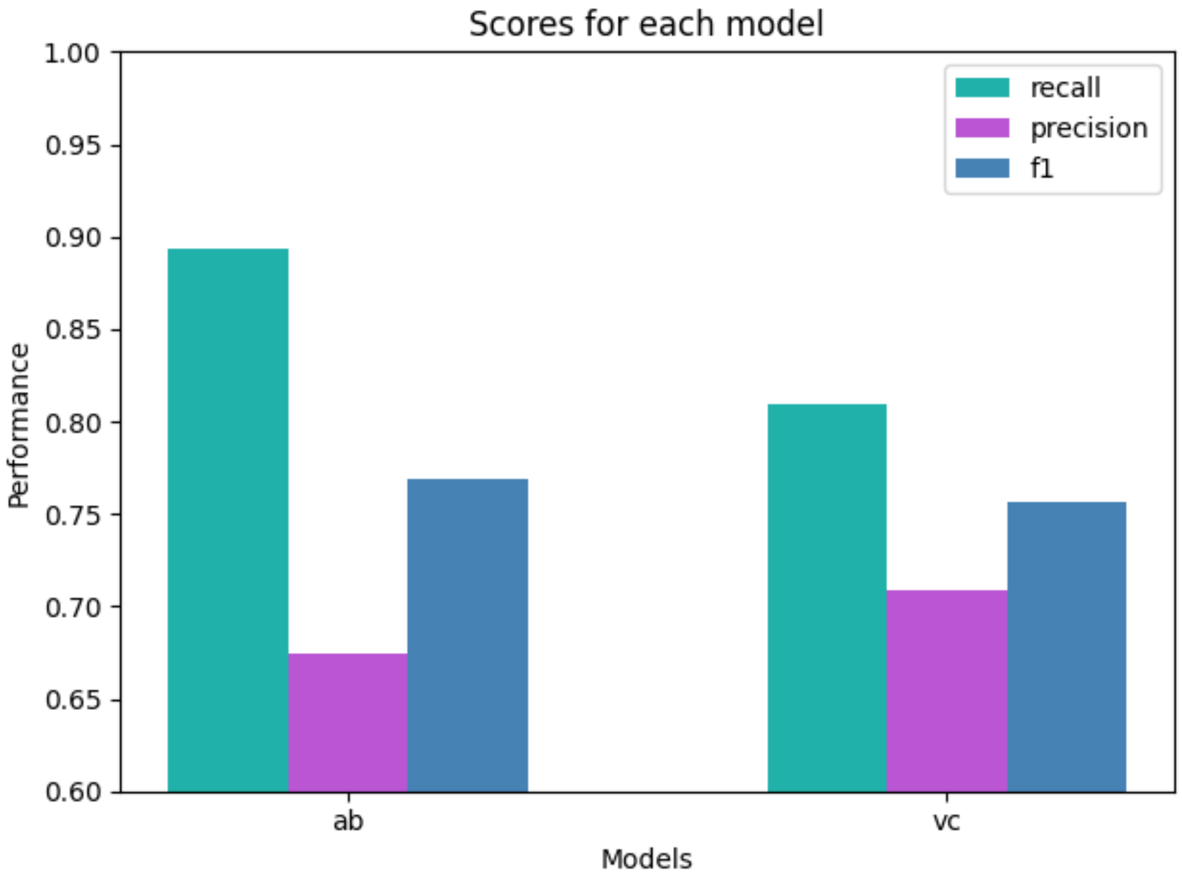
estimator__max_depth	[4, 5, 6, 7, 8]	5
n_estimators	[12, 13, 14, 15]	13
learning_rate	[1.9, 2, 2.1, 2.2]	2.1

learning_rate	[2.05, 2.075, 2.1, 2.125, 2.15]	2.075

learning_rate	[2.0725, 2.075, 2.0775]	2.075

Results (fusion & boosting)

	Recall	Precision	F1
model_vc	0.8095	0.7091	0.7560
model_ab	0.8938	0.6745	0.7688



- AdaBoost (*model_ab*) performs best overall (better tha
  final model
- Tradeoffs regarding computational time and submissio
- Looking back at next steps of last deliverable: all next steps for classifiers implemented, some with more and less success than others. The goal of reaching an f1 score of 80 % was unfortunately not reached.

Tradeoffs: Time vs. Score

SVM

- SVM takes long enough to predict by itself (multiple minutes per iteration)
- To perform grid search with SVM would take days if not weeks
- If unlimited computational time was allowed, we would try Adaboost with SVM
  - This would likely increase F1 score
  - For this project, this does not make sense

  → being allowed to submit an already trained model was communicated very late

  → Earlier communication may have influenced training approaches chosen

Late Fusion with Boosted Model

- Running grid search to find the optimal weights returned a weight of $100\ \%$ for AdaBoost since it had significantly better performance.

→ Late fusion was thus not used in the final predictions