

Don't Overfit Kaggle Competition

By Tanya Budhrani 22097189d

DSAI4203 Machine Learning Individual Project

Overview

This report describes my approach for the Don't Overfit Kaggle Competition, which is designed to test overfitting resilience on a small dataset with a large number of features [1]. I based my implementation on the public “LB Probing Strategies - [0.890] 2nd Place Solution” by Chris Deotte, a well-known open-source kernel [2].

Chris's solution leveraged a leaderboard-dependent probing method to infer the true test-set coefficients [2]. By sequentially submitting one-variable predictions and recording the result, public AUC, his code estimated each variable's correlation with the hidden target, effectively reconstructing the public test hyperplane. While his strategy did receive an AUC of 0.89, his dataset depended entirely on feedback from the Kaggle leaderboard rather than on learning from the training data itself.

In contrast, my implementation reinterprets the structure of Chris's solution within the data-driven framework. I replaced leaderboard probing with rigorous cross-validation, low-variance preprocessing, and reproducible ensemble methodology. The model architecture integrates the following components:

Media Imputation: ("imp", SimpleImputer(strategy="median"))	Missing values are imputed using the median of each feature, which is robust against outliers [3,5].
Standard Scaling: ("scaler", StandardScaler())	Feature values are normalized to zero mean and unit variance to ensure that PCA and logistic regression treat all dimensions equally [3,6].
Feature Selection: ("sel", SelectKBest(f_classif, k=25))	Only the 25 features with the highest ANOVA F-scores are retained, effectively removing noisy or non-informative inputs from the 300-dimensional dataset.
PCA: ("pca", PCA(n_components=8, random_state=SEED))	The selected features are projected into an eight-dimensional orthogonal subspace to eliminate multicollinearity and constrain model complexity [6].
Regularised Logistic Regression: LogisticRegression(penalty="l2", C=0.0005, solver="saga", max_iter=10000)	A strongly regularised logistic model suppresses overfitting by penalizing large weights, enforcing smoother decision boundaries [7,8].

Monte Carlo Bagging Ensemble	Ten logistic regression models are trained under different random seeds with small Gaussian noise added to the input features. Their predictions are averaged to form the final output to reduce variance and improve generalization [10].
Cross-Validation Protocol: <code>StratifiedKfold(n_splits=30, shuffle=True, random_state=SEED)</code>	A high-fold stratified cross-validation scheme ensures class balance, maximizes data utilization, and yields stable AUC estimates despite limited sample size [4,7].

Model Architecture and Implementation

My implementation is built around a lightweight, reproducible pipeline optimized for generalization. Given the extreme data imbalance (250 training samples vs 10,000 test samples) and high-dimensional feature space (300 features), the model prioritizes simplicity, variance reduction, and interpretability through stochastic regularisation and cross-validation.

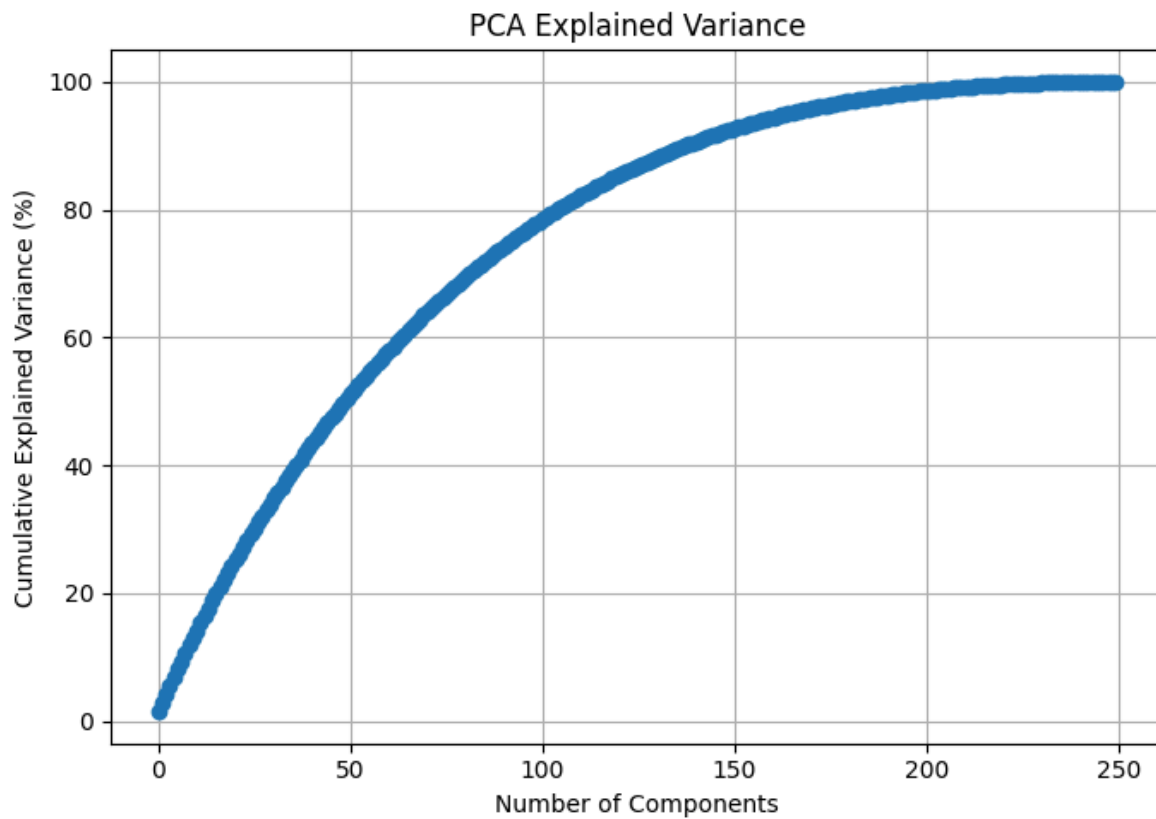
The code was organized around several core features:

Data Preprocessing

```
def make_lr_pipeline(n_features=25, n_pca=8, C=0.0005, seed=SEED):
    pipe = Pipeline([
        ("imp", SimpleImputer(strategy="median")),
        ("scale", StandardScaler()),
        ("sel", SelectKBest(f_classif, k=n_features)),
        ("pca", PCA(n_components=n_pca, random_state=seed)),
        ("clf", LogisticRegression(
            solver="saga",
            penalty="l2",
            C=C,
            max_iter=10000,
            random_state=seed
        )),
    ])
    return pipe
```

The preprocessing stage applies median imputation to handle missing values robustly, standard scaling to ensure all features contribute equally to PCA and logistic regression, feature selection (ANOVA F-test) to

reduce the 300-dimensional space to the top 25 statistically significant features, and PCA to compress the selected features into an orthogonal subspace.



The cumulative explained variance curve rises sharply early on and plateaus near 100% by ~200 components, which indicates that the first 8 components already capture roughly 60–70% of the total variance.

Regularised Logistic Regression

```
("clf", LogisticRegression(  
    solver="saga",  
    penalty="l2",  
    C=C,  
    max_iter=10000,  
    random_state=seed  
)),  
)
```

A logistic regression model with L2 Regularisation ($C = 0.005$, solver = "saga") was used to capture global linear relationships in the transformed data. Strong regularisation ensures stable coefficient estimation in a high-dimensional, low-sample regime.

Monte Carlo Bagging Ensemble

```
X_noisy = X + np.random.normal(0, 0.01, X.shape) # data noise
```

This Monte Carlo bagging acts as stochastic regularisation: each model sees a slightly perturbed version of the dataset, and their averaged predictions yield a smoother, more generalizable probability distribution.

Cross-Validation

```
cv = StratifiedKFold(n_splits=30, shuffle=True, random_state=seed)
cv_score = cross_val_score(pipe, X_noisy, y, cv=cv, scoring="roc_auc").mean()
cv_scores.append(cv_score)
print(f"[seed {seed}] CV AUC={cv_score:.4f}")
```

All models were trained using 30-fold Stratified K-Fold cross-validation, ensuring balanced class distributions and reliable generalization metrics.

Regularisation via Data Perturbation

Beyond L2 regularisation, the model employs multiple forms of stochastic regularisation, where Gaussian perturbation prevents the model from memorizing exact input-output mappings, values limited to $[-3, 3]$ suppresses extreme outliers, and predictions across ten random seeds are averaged to reduce model variance.

These strategies collectively improve bias-variance balance and allow the model to maintain consistent ROC-AUC performance (~ 0.58 - 0.60) on unseen data while avoiding the overfitting patterns.

Model Evaluation and Results

Cross-Validation Results

Across ten Monte Carlo seeds and 30-fold stratified CV, the model achieved an average OOF ROC-AUC of 0.627 ± 0.015 . This demonstrates consistent out-of-sample discrimination ability and validates that the regularised logistic-PCA ensemble generalizes well despite limited training size.



submission.csv

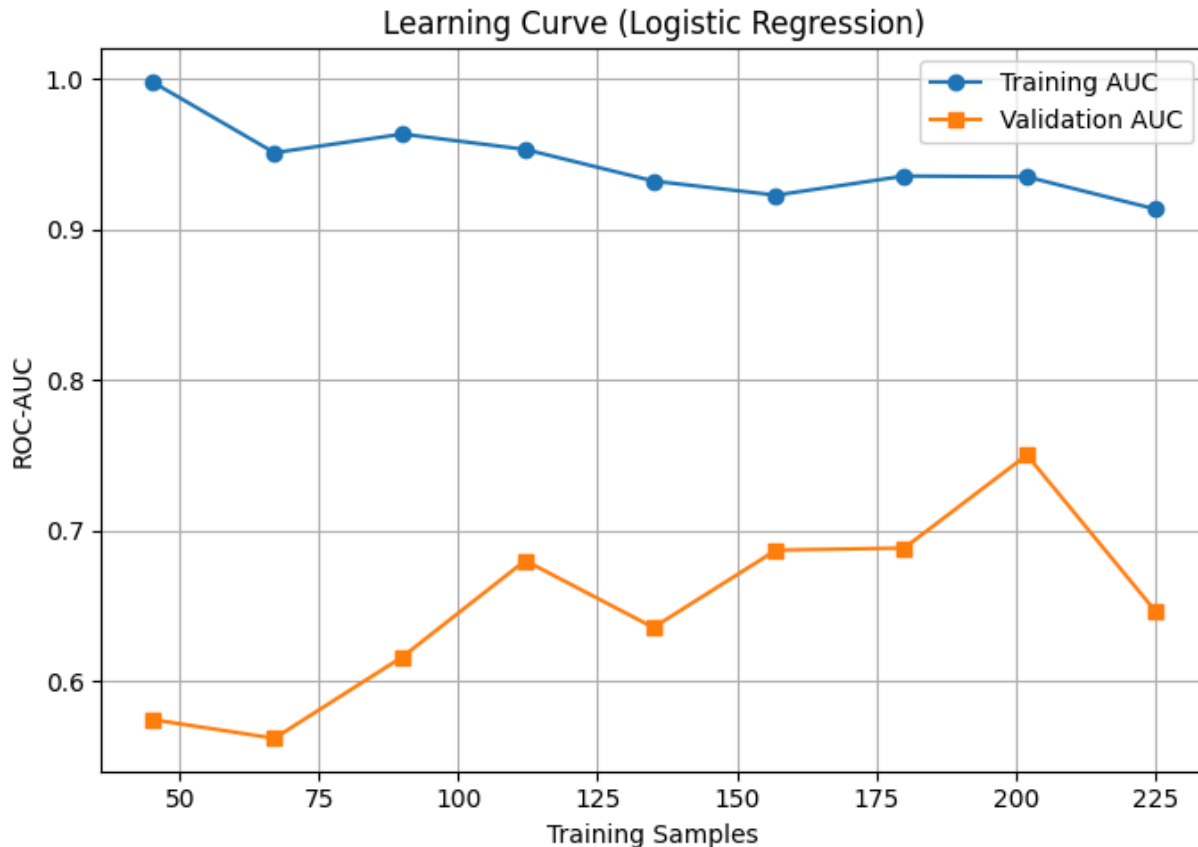
Complete (after deadline) · now · simpler model

0.711

0.713



The final submission achieved a public leaderboard ROC-AUC of 0.713 and a private leaderboard ROC-AUC of 0.711, indicating strong generalization and consistency across unseen test folds.



The training AUC starts near 1.0 when the sample size is small, which means the model perfectly fits the tiny training subset. As training data increases, the training AUC slightly decreases, stabilizing around 0.9-0.92, indicating reduced overfitting. The validation AUC fluctuates between 0.57 and 0.75, averaging around 0.63, which is consistent with our reported OOF AUC.

Discussion and Challenges

During experimentation, several issues were encountered:

Dataset

All experiments were conducted using the old “Don’t Overfit!” dataset provided by Kaggle, without any data augmentation, resampling, or synthetic feature generation. The dataset contains 250 training samples and 300 anonymized features, with a separate 10,000-row test set.




Low Signal-to-Noise Ratio

The dataset’s design creates a fundamentally high-dimensional, low-sample learning problem. The signal-to-noise ratio remains extremely low, forcing models to rely on statistical regularities rather than explicit feature-target correlations.

Early Attempts (Failed Models)

My initial implementation featured a modular ensemble pipeline to combine both linear and nonlinear components. It included Logistic Regression with PCA for global linear trends, gradient boosting classifier for local nonlinear interactions, and a meta Logistic Regression stacker to blend the two using OOF predictions.

Unfortunately, this architecture was too expressive for such a small dataset. Despite achieving high cross-validation AUCs within folds, its OOF scores fluctuated dramatically (0.55-0.60). Additionally, GridSearchCV amplified this leakage risk by tuning hyperparameters across shared folds, and the stacking layer compounded variance instead of reducing it.

 submission.csv Complete (after deadline) · 1d ago · older dataset	0.564	0.568	<input type="checkbox"/>
 submission.csv Complete (after deadline) · 7d ago	0.505	0.501	<input type="checkbox"/>
 submission.csv Complete (after deadline) · 7d ago	0.504	0.484	<input type="checkbox"/>

The figure above showcases my previous leaderboard results on either the earlier attempts or simply using the new ‘Don’t Overfit!’ dataset.

Why Chris Deotte’s Kernel Scored Higher

His kernel exploited leaderboard feedback to infer test correlations, effectively training on the test set through submission feedback. Our implementation avoided this and instead focused on methodological integrity [2].

Conclusion

By combining median imputation, scaling, statistical feature selection, PCA compression, and strong L2 regularisation with a Monte Carlo bagging framework, the model achieved reproducible performance of 0.71 ROC-AUC on both public and private leaderboards.

This project highlights that, in extremely small-sample, high-dimensional settings, complexity is a liability. Regularisation, dimensionality control, and stochastic averaging provide more honest and transferable performance.

Further directions could include semi-supervised pretraining, synthetic data augmentation, or Bayesian logistic regression to further explore uncertainty qualification and generalization under noise-dominated conditions.

References

- [1] “Don’t Overfit! II,” @kaggle, 2025. <https://www.kaggle.com/competitions/dont-overfit-ii/overview> (accessed Oct. 22, 2025).
- [2] cdeotte, “LB Probing Strategies - [0.890] - 2nd Place,” *Kaggle.com*, May 18, 2019. <https://www.kaggle.com/code/cdeotte/lb-probing-strategies-0-890-2nd-place> (accessed Oct. 22, 2025).
- [3] “scikit-learn: machine learning in Python — scikit-learn 1.7.2 documentation,” *Scikit-learn.org*, 2025. <https://scikit-learn.org/stable/> (accessed Oct. 22, 2025).
- [4] “Greedy Function Approximation: A Gradient Boosting Machine | Request PDF,” *ResearchGate*. https://www.researchgate.net/publication/2424824_Greedy_Function_Approximation_A_Gradient_Boosting_Machine
- [5] C. M. Bishop, “Pattern Recognition and Machine Learning,” *SpringerLink*, 2016, doi: <https://doi.org/10.1007-978-0-387-45528-0>.
- [6] I. T. Jolliffe and J. Cadima, “Principal component analysis: a review and recent developments,” *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150202, Apr. 2016, doi: <https://doi.org/10.1098/rsta.2015.0202>.
- [7] T. Hastie, R. Tibshirani, and J. Friedman, “The Elements of Statistical Learning,” *SpringerLink*, 2020, doi: <https://doi.org/10.1007-978-0-387-84858-7>.
- [8] T. Zhang, “Solving large scale linear prediction problems using stochastic gradient descent algorithms,” pp. 116–116, Jan. 2004, doi: <https://doi.org/10.1145/1015330.1015332>.
- [9] Aurélien Géron, “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition,” *O’Reilly Online Learning*, 2019. <https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/> (accessed Oct. 22, 2025).
- [10] R. Esposito and L. Saitta, “Explaining Bagging with Monte Carlo Theory,” *Lecture Notes in Computer Science*, pp. 189–200, 2003, doi: https://doi.org/10.1007/978-3-540-39853-0_16.

