Leveraging dynamic model selection for depression detection

Magnus Sponnich Brørby, Elias Hovdenes & Ísak Páll Gestsson

April 27, 2025

Abstract

In this project, we explore how modern ensemble and model selection techniques can improve classification performance on a synthetic depression survey dataset. Starting from a baseline random forest model, we implemented and evaluated three methods based on recent research: An improved random forest, a SelectionNet and a Greedy Dynamic Feature Selection approach. We also created a combined ensemble by integrating these methods within a SelectionNet structure. Extensive preprocessing and careful dataset splitting were used to avoid data leakage and prepare the data. Our experiments show that while the improvements over the base random forest were modest, the specialized SelectionNet ensemble achieved the best validation accuracy of 93.84%, demonstrating the potential of dynamic model selection even on relatively structured datasets. This project highlights both the opportunities and challenges of applying advanced ensemble techniques to real-world classification problems.

1 Introduction

Ensemble learning is widely used in machine learning, especially for tasks involving structured tabular data. On platforms like Kaggle, some of the best-performing solutions often come from combining several models rather than relying on just one. Random forests, in particular, are known for being strong and reliable out-of-the-box, and they're often hard to beat without some more advanced techniques. That made it a natural place for us to start.

As part of the course INF367A (Applied Machine Learning) at the University of Bergen, we took part in the Kaggle competition *Playground Series - Season 4, Episode 11*, which involved a binary classification problem on a synthetic tabular dataset. The challenge was to accurately classify whether a person was depressed or not based on data from a mental health survey. In the early stages of the project, we tried a few different models like support vector machines, neural networks and decision trees, but the classic random forest consistently gave us the best results. This became our base model and benchmark for the rest of the project.

Later each member of this project picked one recent and relevant research paper and implemented the model described in it. The goal was to improve on the base model and explore more advanced ensemble or selection methods. The three papers we implemented focused on different angles: [1] improving random forests by pruning low-performing and redundant trees, [2] an end-to-end framework (Selection Net) that learns to choose a subset of base learners per sample via a differentiable knapsack layer, and [3] a Greedy Dynamic Feature Selection (GDFS) approach, where a policy network sequentially selects informative features for each instance.

After implementing these models, we also tested a new combination where we used the base random forest, the improved random forest, and GDFS as base learners inside the SelectionNet framework. This allowed the model to select among them dynamically during prediction. The objective was to create a smarter ensemble that could make use of the strengths of each model depending on the input.

In practice, the improvements from the new models and the combined approach were only slightly better than the base random forest, and in some cases, they didn't outperform it at all. Still, this project gave us valuable insight into how ideas from current research can be applied in practice, and it showed us both the strengths and limitations of model selection and ensemble methods when applied to real-world-style datasets.

2 Methods

2.1 Data preprocessing

The dataset used in this work is the synthetic tabular data released for the Kaggle competition. It contained a mix of numerical, categorical, and textual features, along with several missing values and inconsistencies. We performed extensive data exploration and preprocessing to make the data suitable for our models. All raw fields are passed through a single, deterministic pipeline (seeded across Python/NumPy/PyTorch) that performs:

- **Drop `id` & encode `Name` frequency:** We removed the `id` column and encoded `Name` by its frequency in the dataset, assuming name rarity might reflect latent social signals.

- **Binary-encode categorical flags:** The features: `Gender`, `Family History of Mental Illness`, `Have you ever had suicidal thoughts?`, and `Working Professional or Student` were mapped to $\{0,1\}$ using straightforward logical mappings.

- **Impute numeric columns:** For features such as `Academic Pressure`, `Work Pressure`, `CGPA`, and `Financial Stress`, we filled missing values using 0 or the column mean, based on domain context (e.g., students naturally lack work pressure and vice versa).

- **Combine satisfaction:** We observed that `Study Satisfaction` and `Job Satisfaction` were mutually exclusive, so we merged them into a single variable called `Satisfaction`.

- **Ordinal bucketing of categorical features:**
  - `Sleep Duration` was mapped to a 1–4 scale based on sleep quantity bucketing ("5h", ..., "8h").
  - `Dietary Habits` was mapped to $\{0,1,2\}$ representing unhealthy, moderate and healthy respectively.
  - `Degree` values were categorized into educational levels and mapped to $\{0–5\}$ (Other to Doctorate).

- **Profession categorization:** We imputed missing values based on student status and mapped diverse free-text professions into 10 logical categories (e.g., *Technology*, *Education*, *Trade*, etc.), which we then rated based on estimated satisfaction/stress levels.

- **City socio-economics:** We joined external socio-ecconomic city data (population, density, literacy, sex-ratio) for each city and dropped the original `City` column.

- **Multiple training and validation sets:** The dataset provided by the competition was first split into two parts: a training set and a validation set, with an 80% training split. For simplicity, lets just call these two sets for `train1` and `val1`. These were used for training and validating all models, except for the base learners used within the SelectionNet ensembles. To correctly train the base learners for SelectionNet without causing data leakage, we further split the original training set (`train1`) into two new subsets: `train2` and `val2`. The base learners inside SelectionNet were trained on `train2` and validated on `val2`, ensuring that their performance estimates were based on unseen data. Once the base learners were trained, the SelectionNet itself was trained on the full `train1` set and validated on `val1`, aligning the validation strategy with the other models. This careful separation of datasets was crucial for a fair evaluation and for preventing any information leakage into the model selection process.

- **Normalizing the dataset** Since our project involved both neural networks and random forests, we decided to normalize the dataset to ensure compatibility across all models. Neural networks generally require normalized or standardized input features to train efficiently and achieve good performance, as varying feature scales can negatively impact gradient-based optimization. Random forests, on the other hand, are scale-invariant and do not rely on feature magnitudes when making splits. Therefore, normalization was a safe and effective preprocessing step that improved the training dynamics of the neural networks without affecting the performance of the random forest models.

2.2 Classic Random Forest (Base Model)

After testing several learning algorithms on the dataset, the classic `RandomForestClassifier` from `sklearn.ensemble` consistently achieved the highest performance among our baseline models. To optimize its performance, we conducted a hyperparameter search over the following ranges:

- `max_depth`: [10, 20, 30]
- `min_samples_leaf`: [2, 4]
- `min_samples_split`: [2, 4]
- `n_estimators`: [50, 100, 200]

The best configuration found for this dataset is reported in Appendix C. We used this model as the benchmark for evaluating the improvements achieved by the models developed later in the project.

2.3 Selection Net

We implement the end-to-end Combinatorial Ensemble Learning (e2e-CEL) framework of Kotary *et al.* [2] to dynamically pick a small subset of our depression classifiers at inference time.

- **Preprocess & specialized learners:** We form two "specialized" datasets: for each target value $t \in \{0, 1\}$, we take *all* samples with label t and *randomly sample* 5% of the opposite class via

```
specialized = data[data.Depression ==
↪  t]
augmented  = data[data.Depression !=
↪  t].sample(frac=0.05,
↪  random_state=seed)
combined   = pd.concat([specialized,
↪  augmented], ignore_index=True)
```

This yields two 95/5 splits (depressed-majority and non-depressed-majority). On each split we train five models (LogisticRegression, DecisionTreeClassifier, RandomForestClassifier, XGBClassifier, AdaBoostClassifier) via 5-fold CV with grid search, for a total of 10 specialized learners (see Appendix B for code and exact grids).

- **Selection network & differentiable knapsack:** We use a two-layer MLP

$$\text{SelectionNetwork}(F, N):$$
$$F \xrightarrow{\text{Linear}} 128 \xrightarrow{\text{ReLU}} N$$

where F is the feature dimension and $N = 10$ base learners. Its raw scores $\hat{c} \in \mathbb{R}^N$ are L2-normalized and fed into a Monte Carlo–smoothed top-k knapsack layer ($\epsilon = 0.1$, $m = 1000$), as detailed in Kotary *et al.* [2]. Selected learners' softmax outputs are then masked, summed, and re-softmaxed to yield the final ensemble probabilities.

- **Tuning k:** We evaluate $k \in \{1, \ldots, 10\}$, training for 5 epochs each (batch size = 512, Adam lr = 1e-3, CrossEntropyLoss), with $\epsilon = 0.1$ and $m = 1000$. We record training loss and validation accuracy per epoch, choose the best k, and save its state-dict.

2.4 Improved Random Forest

Since our best-performing base model was the standard random forest, we decided to explore methods that could enhance its performance further. This led us to the paper by Sun et al. [1], which proposes a method for selecting decision trees based on both accuracy and pairwise correlation.

The approach starts by training an $N_{\text{Final}} + m \times N_{\text{Final}}$ large pool of decision trees using bagging. Each tree is evaluated on a reserved validation dataset that is split into three parts, and the average classification accuracy across these splits is used as the tree's performance score.

To reduce redundancy in the ensemble, the method introduces an improved correlation measurement based on the cosine similarity of the trees' feature subsets. By treating each feature subset as a vector, the angle between pairs of trees can be calculated. The grid search method is then used to find an optimal inner product (or angle) threshold that determines which pairs of trees are too correlated. Among each pair that is too correlated, the tree with the lower average accuracy is marked for deletion. The trees with the lowest average accuracy among those marked as deletable are pruned sequentially until the desired number of trees remains. If the target number of trees is not reached — that is, if too few trees were initially marked for deletion — the pruning process continues by removing additional trees based solely on their accuracy scores, regardless of their deletion status. This continues until the final ensemble size N_{final} is achieved.

When tuning the inner product threshold for correlation filtering, it was important to select threshold values that were meaningful for the current set of trees. Instead of using a fixed set of thresholds for all models, we based the candidate thresholds on the distribution of pairwise correlation values within each ensemble. This approach avoids tuning on thresholds that are either too large or too small compared to the actual correlations, which would otherwise waste computational resources and provide little useful information.

Specifically, we computed the mean and minimum correlation values among all tree pairs and constructed five candidate thresholds by linearly interpolating between the minimum, mean, and maximum correlation values. This ensured that the tuning process focused on thresholds relevant to the specific correlation structure of each model.

Constructing the best possible Improved Random Forest involved hypertuning on these parameters:

- N_final = [100, 200]
- m = [0.3, 0.7]
- max_depth = [8, 15]
- min_samples_leaf = [2, 4]
- min_samples_split = [2, 4]

The best configuration found for this dataset is reported in Appendix C.

2.5 Greedy Dynamic Feature Selection

With many potential features and with the idea in mind that some features like socio-economic status or history of mental illness have to have a much bigger impact on an illness such as depression we found that the paper by Covert et al. [3], could help us in determining which features count the most. This method considers dynamic feature selection where the goal is to pick the most informative features. This is done by training two neural networks. One policy network that greedily selects the features that give us the most information about the response variable. And one selection network that makes prediction with the given feature set. We opted for a similar approach as the authors where two identical fully connected nets are used. The hyperparameter values for the net and training is given in table C. Then an input mask is created and applied to the input features. Afterwards the selecting network for each dataset is pretrained with 100 epochs and then the combined selector and policy network is trained on each dataset with cross entropy loss function for 3 epochs for every temperature parameter τ as is explained in [3]. The open-source code of the users is given here.

2.6 Combined Ensemble

To leverage the strengths of each individual method, we build a single "smart" ensemble by using Greedy Dynamic Feature Selection, our Improved Random Forest, and the original base model as the three families of base learners inside the Selection Net framework. For each family, we train three variants (on the full training set for base learners, a 95% depressed–enriched set, and a 95% non-depressed–enriched set), yielding nine total pre-trained predictors. These nine models' soft-probability outputs are then fed into our two-layer MLP + Knapsack layer ($\varepsilon = 0.1$, $m = 1000$), which learns via end-to-end differentiable top-k selection how to pick and weight the best subset of learners for each input. Finally, we tune for $k \in \{1, \ldots, 9\}$ (5 epochs, batch size 512, Adam 1×10^{-3}) on validation to choose the optimal sub-ensemble size and corresponding network weights.

3 Results

The results for the different models are given in table 1. The best performing model with a validation accuracy of 93.84% percent was the specialized Selection Net. That model was then used to make predictions on the test data and there it posted a **private score of 93.912%** where private score contains 80% of the test data. The mean validation accuracy of the models is 93.72% with a 0.0007 margin of error at 95% confidence interval.

Model name	Val accuracy
Selection Net special	**0.9384**
Selection Net Combined	0.9376
Improved random forest	0.9361
GDFS	0.9372
Base model random forest	0.9370

Table 1: Results of the different models

The optimal hyperparameter configurations for each model (Random Forest, Improved Random Forest, Selection Net specialized, Greedy Dynamic Feature Selection, and Combined Selection Net) are provided in Appendix C.

4 Discussion

Interpretation of results. Our experiments demonstrate that the *Selection Net* specialized ensemble achieved the highest validation accuracy (93.84 %), narrowly outperforming both the Combined Selection Net (93.76 %) and the baseline Random Forest (93.70 %). This confirms that end-to-end, differentiable model selection can yield small but consistent gains over homogeneous or manually tuned ensembles on this synthetic depression dataset.

Achievement of project objective. Our aim was to create a more powerful and flexible ensemble by first implementing three distinct methods (pruned Random Forests, Greedy Dynamic Feature Selection, and a generic Selection Net) and then combining them into a single pipeline. Surprisingly, the specialized Selection Net, which used its own arbitrary set of ten base learners instead of those derived from the pruned forests or GDFS models, achieved the highest validation accuracy (93.84 %), outperforming both the baseline Random Forest (93.70 %) and the Combined Selection Net that included the pruned forests and GDFS as base learners. This result shows that the Selection Net's per sample selection mechanism alone can effectively leverage diverse predictive signals even without directly including the other advanced methods.

Comparison to other competition approaches. Leading entries in the Kaggle *Playground Series – Season 4, Episode 11* contest relied heavily on static ensembles of gradient-boosted trees (XGBoost, LightGBM) and manual feature engineering, achieving winning scores near 94.2 %. In contrast, our method adapts per input by selecting a small subset of specialized learners. Although our top score (93.912 % private) is slightly below the contest leaders, we accomplish this with minimal manual feature work and by leveraging end-to-end, differentiable selection.

Limitations and future work. A major challenge was the computational cost of the Monte Carlo knapsack layer ($\epsilon = 0.1$, $m = 1000$). Each forward/backward pass requires a thousand top-k selections per sample, which limited our ability to explore longer training schedules. Empirically, we observed diminishing returns beyond $m = 1000$ and five epochs per k, so we constrained our search to balance accuracy and runtime. Future work could investigate more efficient gradient estimators (e.g., Gumbel–Softmax relaxations) or adaptive sampling schemes to reduce training time without sacrificing model fidelity.

The Greedy dynamic feature selection (GDFS) could be improved upon by conducting a hyperparameter search and tuning. That was not done because of the computational cost of performing such a search and the time it would take. Better results from the GDFS could also be obtained by training it for more epochs than only 3 per temperature value but we opted not to in order to save computational resources. That would fit the model better to the data and give it more chance to capture the underlying patterns in the data. Even though fewer epochs reduce the chance of overfitting we believe the drop out rate in the model would help regulate that. So given more time and resources we would allocate more training epochs to the GDFS.

Another limitation is that our pipeline and specialized data splits were tuned exclusively on a synthetic Kaggle competition dataset. While our deterministic seeding ensures reproducibility, real-world clinical depression data may exhibit different feature distributions and noise characteristics. Therefore, external validation on clinically collected cohorts would be necessary to assess generalization.

5 Conclusion

In summary, the Selection Net framework effectively leverages heterogeneous base models to improve binary classification accuracy with only modest overhead. By combining carefully constructed specialized splits, exhaustive hyperparameter searches (see Appendix B), and a Monte Carlo–smoothed knapsack layer, we provide a reproducible recipe for dynamic model selection.

By integrating machine learning methodologies into the mental health domain this project underscores the potential of computational tools to contribute to healthcare solutions. While the gains over the baseline were modest, this work lays a foundation for applying advanced ensemble learning frameworks to mental health research.

References

[1] Z. Sun, G. Wang, P. Li, H. Wang, M. Zhang, and X. Liang, "An improved random forest based on the classification accuracy and correlation measurement of decision trees," *Expert Systems with Applications*, p. 19, 2024.

[2] J. Kotary, V. Di Vito, and F. Fioretto, "Differentiable model selection for ensemble learning," *arXiv preprint arXiv:2211.00251v2*, 2023.

[3] I. Covert, W. Qiu, M. Lu, N. Kim, N. White, and S.-I. Lee, "Learning to maximize mutual information for dynamic feature selection," 2023.

A Appendix

B Hyperparameter Configurations for Specialized Base Learners

Model Imports and Grid Definitions

```
from sklearn.linear_model    import LogisticRegression
from sklearn.tree            import DecisionTreeClassifier
from sklearn.ensemble        import RandomForestClassifier, AdaBoostClassifier
import xgboost                as xgb
from sklearn.model_selection import GridSearchCV
```

Grid-Search Summary

Table 2: Grid-search configurations for specialized base learners (cv=5, scoring='accuracy')

Model	**Init. kwargs**	**Param. grid**
LogisticRegression	max_iter=1000	C $\in \{0.01, 0.1, 1, 10\}$
DecisionTreeClassifier	random_state=seed	max_depth$\in \{None, 5, 10\}$, min_samples_split$\in \{2, 5, 10\}$
RandomForestClassifier	random_state=seed	n_estimators$\in \{100, 200\}$, max_depth$\in \{None, 10, 20\}$
XGBClassifier	eval_metric='logloss'	learning_rate$\in \{0.01, 0.1, 0.2\}$, max_depth$\in \{3, 5, 7\}$, n_estimators$\in \{100, 200\}$
AdaBoostClassifier	algorithm='SAMME', random_state=seed	n_estimators$\in \{50, 100, 200\}$, learning_rate$\in \{0.5, 1.0, 1.5\}$

C Best Hyperparameter Configurations

Below we list the final hyperparameter settings for each of the five methods evaluated in this paper.

Base Random Forest

Hyperparameter	Value
max_depth	30
min_samples_leaf	2
min_samples_split	2
n_estimators	200

Table 3: Best hyperparameters for the baseline Random Forest.

Improved Random Forest

Hyperparameter	Value
N_{final}	200
m	0.7
max_depth	15
min_samples_leaf	4
min_samples_split	2

Table 4: Best hyperparameters for the Improved Random Forest.

Selection Net (Specialized)

Hyperparameter	Value
top-k	2

Table 5: Best top-k value for the specialized Selection Net.

Learner	**Best Hyperparameters**
logistic_depr	{'C': 10}
tree_depr	{'max_depth': 5, 'min_samples_split': 2}
rf_depr	{'max_depth': None, 'n_estimators': 100}
xgb_depr	{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200}
ada_depr	{'learning_rate': 1.5, 'n_estimators': 200}
logistic_nondepr	{'C': 1}
tree_nondepr	{'max_depth': 5, 'min_samples_split': 2}
rf_nondepr	{'max_depth': 20, 'n_estimators': 200}
xgb_nondepr	{'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200}
ada_nondepr	{'learning_rate': 1.5, 'n_estimators': 200}

Table 6: Best hyperparameter configurations for each of the 10 specialized base learners.

Greedy Dynamic Feature Selection

Hyperparameter	Value
Batch size	128
Dropout rate	0.4
Hidden layer size	64
Learning rate	0.001
Max features	20

Table 7: Hyperparameters for the Greedy Dynamic Feature Selection method.

Combined Selection Net

Hyperparameter	Value
top-k	8

Table 8: Best top-k value for the Combined Selection Net.

Table 9: Best hyperparameters for each base learner in the Combined Selection Net.

Learner	**Best Hyperparameters**
Improved RF (full)	{N_final=200, m=0.7, max_depth=15, min_samples_leaf=4, min_samples_split=2}
Improved RF (depressed)	{N_final=200, m=0.7, max_depth=15, min_samples_leaf=2, min_samples_split=2}
Improved RF (happy)	{N_final=200, m=0.7, max_depth=15, min_samples_leaf=4, min_samples_split=2}
Base RF (full)	{max_depth=20, min_samples_leaf=2, min_samples_split=2, n_estimators=200}
Base RF (depressed)	{max_depth=20, min_samples_leaf=2, min_samples_split=2, n_estimators=200}
Base RF (happy)	{max_depth=20, min_samples_leaf=2, min_samples_split=2, n_estimators=200}
Greedy DFS (full)	see Table 7
Greedy DFS (depressed)	see Table 7
Greedy DFS (happy)	see Table 7