ID2214
Activity Prediction for Chemical Compounds Using Machine Learning

Adalet Adiljan
Group 15

December 2024

1 Introduction

The goal with this project is to develop two prediction models that classifies 69,646 instances of chemical compounds as active or inactive based on choice of features. Given training dataset consisting of 208 938 instances ad the test (unseen) dataset with 69 646 instances, we are aimed to find the best suitable prediction model that focuses on maximizing area under the ROC curve (AUC). The model with best performance is then aimed to predict the probabilities for the 'ACTIVE' labels, for each and every instance. To reach this stage, a series of essential preprocessing and modeling steps must be performed to ensure the accuracy and reliability of the final model. In our case, the important steps have included feature engineering, class imbalance correction, hyper-parameter tuning and best model evaluation.

2 Methodology

The prior knowledge of data preparation techniques from past assignments, YouTube tutorials, lecture materials, and articles outlining step-by-step processes played a significant role in achieving the goals of this project.

2.1 Data Preparation (Feature engineering)

Before starting the project, we needed to ensure that the training dataset is free of missing or incorrect values and that we understand the names of all columns and rows. After inspecting the name of the columns, and the number of total instances, the relevant features extracted including: Molecular Descriptors (Molecular Weight, TPSA, and LogP), Lipinski's Rule of Five features, Fragment Features, and Morgan Fingerprints. The rationale behind these choice of features was to make sure to include all the features that might correlate with

the compounds likehood of being active. The libraries used for this mapping are RDKit, Pandas, and NumPy. The same feature extraction was applied for both the training_smiles.csv and test_smiles.csv to ensure that they are compatible to eachother. The extracted features were then saved in pickle to be accessed and be served as inputs for next step in the process.

```
Loaded Processed Training Data:
   INDEX                                             SMILES  ACTIVE  \
0      1  CCOc1cc(CNc2cccc3ccccc23)cc(Br)c1OCC(=O)NC(C)(C)C       0
1      2  COc1cc(-c2ccc(Nc3ccccc3C(=O)O)c(OC)c2)ccc1Nc1c...       0
2      3               O=c1cc(O)n(-c2ccc(Cl)cc2)c(=S)[nH]1       0
3      4    COc1ccc(NC(=O)C/C(C)=N/NC(=O)CCc2ccccc2)c(OC)c1       0
4      5           CC(=O)/C=C(\C)NCCNc1c(F)c(F)c(CO)c(F)c1F       0

   exact_mol_weight    tpsa     LogP  heavy_atom_count  rotatable_bonds  \
0       484.136155   59.59  5.90660                31                8
1       484.163436  117.12  6.25440                36                9
2       253.991676   58.02  2.25409                16                1
3       383.184506   89.02  3.15730                28                9
4       320.114791   61.36  2.22960                22                7

   hydrogen_bond_donors  hydrogen_bond_acceptors  ...  fp_1014  fp_1015  \
0                     2                        4  ...        0        0
1                     4                        6  ...        0        0
2                     2                        4  ...        0        0
3                     2                        5  ...        0        0
4                     3                        4  ...        0        0

   fp_1016  fp_1017  fp_1018  fp_1019  fp_1020  fp_1021  fp_1022  fp_1023
0        0        0        0        0        0        0        0        0
1        0        0        0        1        0        0        0        0
2        0        0        0        0        0        0        0        0
3        0        1        0        1        0        0        0        0
4        0        1        0        0        0        0        0        0
```

Figure 1: Final dataset of chemical compounds with extracted features.

Each row to different instance of chemical compounds, identified by the 'INDEX' and 'SMILES' column. The 'ACTIVE' column corresponds to the given instance being acive (1) or inactive (0), along with rest of the columns that are the extracted features and fingerprints of 1024-bit fingerprint vectors.

2.2 Addressing Class Imbalance

Given the article [1], the data preparation stage is described as an ongoing process rather than a one-time task. This was highlighted upon realizing a significant class imbalance between the 0 and 1 values, large portion of inactive compounds compared to negative ones, given the image below.

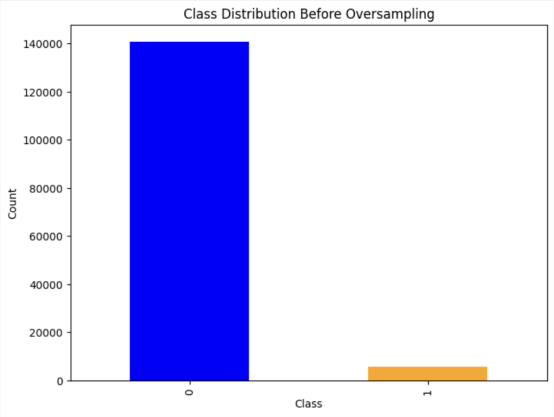

Figure 2: Class distribution of 1:es and 0:es before oversampling, where blue bar represents total instances of inactive compounds, while orange bar number of active compounds.

To resolve this, RandomOverSampler from the imbalanced-learn (imblearn) library was used. Random Oversampling was applied to the training set to ensure equal distribution of these. If this wouldn't be applied, this would increase the risk of the model being biased towards the majority class. The 'ACTIVE' column serves as the target variable, extracted as 'y'. Oversampling was implemented given the following code:

```
# Split features and target variable
X = processed_train_df.drop(columns=['SMILES', 'ACTIVE', 'INDEX'])  # Features
y = processed_train_df['ACTIVE']  # Target variable
X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size = 0.30,
stratify=y, random_state=42)

oversampler = RandomOverSampler
(sampling_strategy='auto', random_state=42)
X_train_resampled, y_train_resampled = oversampler
.fit_resample(X_train, y_train)
```

Image below shows the class balance after applying oversampling, where the minority class will be oversampled to match the larger class. Oversampling in this case could be a more preferable option instead of undersampling, since we want to preserve as much data as possible without affecting the complex relation it has with other features. Another reason to why oversampling could be a better option is due to our high dimensional data. It enables the model to learn from a "richer" dataset which in turn improves its ability to generalize to new instances.

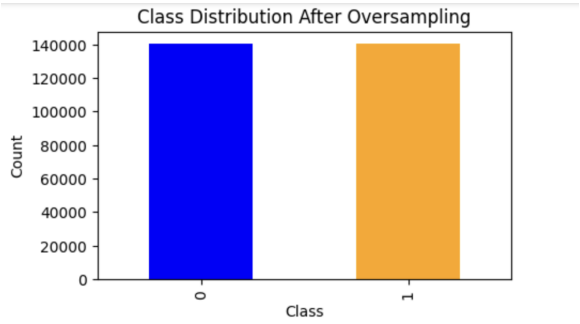

Figure 3: Balanced class after oversampling

Before oversampling, the data set is split into 70% training and 30% testing using the split method from scikit-learn to preserve the original proportion in both sets. Oversampling is then applied to the training set to avoid overly optimistic results and realistic model performance. In preparing the data, two distinct representations were were considered and evaluated:

1. Full Feature Representation: This representation evolved extracting molecular descriptors (such as molecular weight, TPSA, LogP, etc.) along with the corresponding fingerprint vectors for each chemical compound instance. No dimensionality reduction was applied in this case, since I wanted to see how the fingerprints (not reduced) and the features affects the prediction performance.

2. PCA-Reduced Representation: In this approach, Principal Component Analysis (PCA) was applied to the fingerprint features to reduce dimensionality, while the extracted molecular descriptors were retained in their original form. The aim was to reduce the sparsity and high dimensionality associated with the fingerprints. This was observed during the data preparation phase, while exploring and analyzing the dataset. It became evident that 1024 fingerprints features was significantly larger compared to the other features, resulting in a large feature space. Given the fingerprint values, which consisted of varying zeros and ones, it was also observed that sparsity could contribute to the potential curse of dimensionality [2]. Therefore, I wanted to see how reduced dimensionality of the fingerprints would affect the predicting performance. The average sparsity of the fingerprint features was calculated by considering the total instances of 0:es for each fingerprint feature:

```
sparsity = (processed_train_df[fingerprint_columns]==0)
.mean().mean() * 100
print(f"Average sparsity of fingerprint features: {sparsity:.2f}%")
```

By going through each element in the fingerprints, and counting all zeroes and find that the total amount of zeroes is greater than half the elements, then the fingerprint features would be considered sparse.[8] The printed result gave us that the fingerprints were 95,67% sparse, corresponding to 95% of these features were zeros (corresponding to absence in the chemical compound).

2.3 External Tools and Libraries Used

This section summarizes all the libraries used, sorted for each purpose and stages:

Data Preprocessing and Feature Extraction

- **Pandas** – Used for loading, manipulating, and converting features to categorical types.
- **RDKit & Chem** – The Chem toolkit was valuable for generating molecular descriptors and Morgan fingerprints.
- **Pickle** – Used for saving and loading trained machine learning models, to avoid the need for retraining.

Class Imbalance

- **imblearn.over_sampling.RandomOverSampler** – Applied fit_resample to oversample the minority class and resolve class imbalance for the 'ACTIVE' label.

Machine Learning Models

- **sklearn.ensemble.RandomForestClassifier** – Inbuilt random forest, utilizing ensemble learning methods.
- **lightgbm.LGBMClassifier** – Gradient boosting framework optimized for large datasets with improved speed and efficiency.

Model Evaluation

- **sklearn.metrics** – Provided inbuilt functions for roc_auc_score, classification_report, and accuracy_score.
- **sklearn.model_selection** – Offered performance evaluation through train_test_split.

Dimensionality Reduction

- **sklearn.decomposition.PCA** – Utilized to reduce dimensionality and address sparsity. The functions fit_transform and transform were applied.

Data Visualization

- **matplotlib.pyplot** – Used to visualize class distribution before and after oversampling and plot explained variance for PCA.

2.4 Representation of Data, PCA

Representing the fingerprint features as principal components means to create new variables that are combinations of the original features, with the aim to capture the most important patterns in the data. [2], meaning, to reduce them in a lower dimensional space while discarding noisy (non-informative) ones.

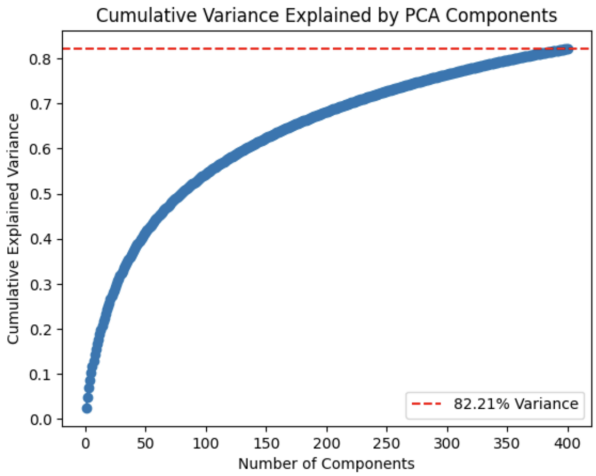

Figure 4: Cumulative explained variance as a function of the number of PCA components. The red dashed line marks the threshold at 82.21% variance with 400 components needed to capture this level.

80% variance is often considered sufficient and going closer to 100% variance would correspond closer to 1000 components given the second image. It would defeat the purpose of dimensionality reduction, therefore the reasoning behind this was to balance variance retention with dimensionality reduction.

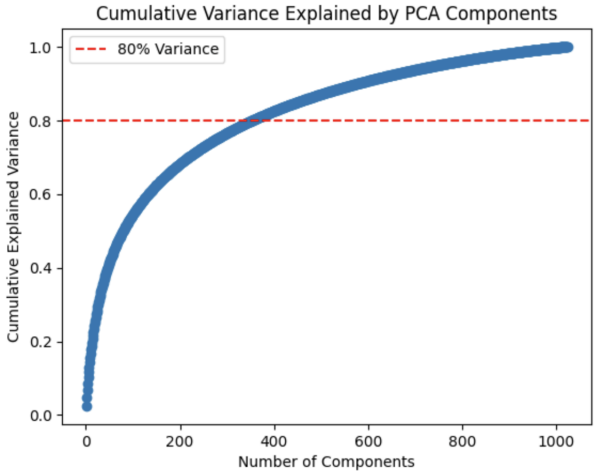

Figure 5: Cumulative explained variance when considering all the fingerprint features.

3 Learning Algorithms and Model Development

3.1 Machine Learning Models

Two models of choice was the inbuilt learning algorithms Random Forest (RandomForestClassifier) from sklearn and LGBMClassifier from lightGBM library. The 'ACTIVE' column was dropped to be used as the target variable for training the models to recognize active or inactive compounds based on the different representations of the dataset.

3.1.1 Random Forest

For the first representation, the model was trained for the resampled training set:

```
    model = RandomForestClassifier(
    n_estimators=100,
    max_depth=None,
    min_samples_split=2,
    min_samples_leaf=1,
    random_state=42
)
model.fit(X_train_resampled, y_train_resampled)
y_proba = model.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, y_proba)
print(f"Random Forest ROC AUC Score, test data: {roc_auc:.10f}")
```

The ROC AUC score and classification report was calculated by the unseen 30% test data using sklearn's inbuilt library for the AUC score, called roc_auc_score and classification_report. Setting the parameters manually values gave us the highest result of Random Forest ROC AUC Score: 0.9138931524 = 0.91. The chosen hyperparameters for the Random Forest model were largely set to their default values. 'n_estimators' was set to 100, representing the number of trees in the forest. 'max_depth' the maximum depth of the tree with the default value 'None', 'min_samples_split & 'min_samples_leaf' set with their default values [6]. The random_state was initially assigned an arbitrary value (42) for convenience during experimentation.

3.1.2 LightGBM

During the initial experimantion phase of trying with different models, such as traditional gradient boosting and support vector machines, these models exhibited significant computational and time complexity for the training times. The weaknesses drove me to find a better suitable model, leading me to LightGBM that has optimized training speed and low memory consumption. Another reason to why the model was suitable, was its inherent ability to handle high-dimensional data and sparse features. [3]

```
lgbm_model = LGBMClassifier(n_estimators = 200, learning_rate=0.1,
random_state=42)
lgbm_model.fit(X_train_resampled, y_train_resampled)

y_proba_lgbm = lgbm_model.predict_proba(X_test)[:, 1]
y_pred_lgbm = lgbm_model.predict(X_test)

roc_auc_lgbm = roc_auc_score(y_test, y_proba_lgbm)
print(f"LightGBM ROC AUC Score for the test set: {roc_auc_lgbm:.10f}")
```

The way of defining the hyperparameters for ligthGBM can be applied following this guide [4], where the set value for these was carefully chosen to balance model complexity and training time. The definition of parameters are following: 'n_estimators' stands for boosting rounds (number of boosted trees to fit), 'learning_rate for boosting learning rate, and 'random_state for controlling the randomness in model training. [5]. Given the documentation, the default set value for estimators is 100, but a value of 200 was moderate to ensure sufficient complexity to ensure the decreased risk of overfitting and prolonged training time. The learning rate of 0.1 (default) was retained to manage a decent training time and also to match the number of estimators, since low learning rate of 0.01, 0.05 etc requires more estimators to achieve a good performance. [9]

The random_state was arbitrarily set to ensure reproducibility. This combination of parameters resulted in a LightGBM ROC AUC Score of 0.8947124613 = 0.89 for the test set.

When it comes to the second representation, the order of operations is significant to prevent data leakage for the train-test sample. we start off by selecting all the columns starting with 'fp_' that are isolated for PCA. The isolated fingerprints is then retrieved from the trained and test set from the split resampled data. Further, the PCA transformed dataset is then concatenated with the remaining features.

```
fingerprint_columns = [col for col in processed_train_df.columns if
col.startswith('fp_')]
pca = PCA(n_components=400, random_state=42)
X_train_pca = pca.fit_transform(X_train_resampled[fingerprint_columns])
X_test_pca = pca.transform(X_test[fingerprint_columns])
(:)
X_train_rest = X_train_resampled.drop(columns=fingerprint_columns)
X_test_rest = X_test.drop(columns=fingerprint_columns)
```

The ROC AUC score was computed as following:

```
y_proba = lgbm_model.predict_proba(X_test_combined)[:, 1]
print(f"LightGBM ROC AUC: {roc_auc_score(y_test, y_proba):.4f}")
-----------------------------------------------------
y_proba_rf = rf_model.predict_proba(X_test_combined)[:, 1]
print(f"Random Forest ROC AUC: {roc_auc_score(y_test, y_proba_rf):.4f}")
```

This approach resulted in AUC scores of 0.87 for LightGBM and 0.85 for Random Forest. The AUC is calculated using the original test set and the predicted probabilities for 1:es for each model. In both cases, X_test_combined consists of original descriptors concatenated with the PCA-transformed fingerprint features.

3.2 Results & Findings

After evaluating the final results, it was evident that the Random Forest model trained on the full feature set (without PCA) achieved the highest AUC score of 0.91. The best prediction model was then applied to the unseen smiles.csv test dataset. Prior to this, the test data went through the same feature extraction process as the training data. Except for the feature extraction, all non-essential columns, such as 'SMILES' strings and 'INDEX', were excluded. After applying the same preprocessing steps to the test data, the predicted probabilities.

3.3 Performance Metrics

This chapter of the report consists of classification reports for the models performance from sklearn (scikit-learn) library. It presents other informative insights about the models performance for predicting both the active and inactive values. The collected insights and takeaways from the report is that Random Forest performs the best in full feature space but struggling with PCA, while lightGBM indicates its adaptability to dimensionality reduction and sparsity, outperforms RF in PCA-transformed data.

3.3.1 Random Forest

Random Forest Classification Report:

	precision	recall	f1-score	support
0	0.98	1.00	0.99	60302
1	0.77	0.40	0.53	2380
accuracy			0.97	62682
macro avg	0.88	0.70	0.76	62682
weighted avg	0.97	0.97	0.97	62682

3.3.2 LightGBM

LightGBM Classification Report:

	precision	recall	f1-score	support
0	0.99	0.90	0.94	60302
1	0.22	0.71	0.34	2380

```
    accuracy                           0.89     62682
   macro avg       0.60      0.81      0.64     62682
weighted avg       0.96      0.89      0.92     62682
```

3.3.3 Random Forest for PCA transformed features

```
Random Forest ROC AUC: 0.8536
              precision    recall  f1-score   support

           0       0.96      1.00      0.98     60302
           1       0.82      0.05      0.09      2380

    accuracy                           0.96     62682
   macro avg       0.89      0.52      0.54     62682
weighted avg       0.96      0.96      0.95     62682
```

3.3.4 LightGBM for PCA transformed features

```
LightGBM ROC AUC: 0.8718
              precision    recall  f1-score   support

           0       0.98      0.93      0.96     60302
           1       0.26      0.61      0.36      2380

    accuracy                           0.92     62682
   macro avg       0.62      0.77      0.66     62682
weighted avg       0.96      0.92      0.93     62682
```

4 Discussion & Analysis

After expecting to see a better performance of the machine learning models, I was taken by surprise when I realize that both of them didn't perform well for the PCA reduced dataset. The reason for this could be due to slight loss of critical information since while its trying to preserve the variance may discard low-variance features that could contain important information for rare occasions that label is 1.0 (active). Another reason for the weaker performance could be related to the complex relationship of the non-linearity between the features and the target variable. The reason to why RF performs better without PCA reduction could highlight the models dependency on individual feature importance. Therefore, transforming them into components could cause the Random Forest to not interpret it well. Looking at the Recall for class 1, we can see that it drops drastically to 5% after applying PCA. In contrast, LightGBM maintains a higher recall of 61% for the PCA applied dataset which indicates a better performance. This can be explained by how lightGBM is engineered to handle high dimensional data and sparse features. Compared to Random

Forest, it only expands the nodes that minimizes the loss. It also has integrated functions like GOSS and EFB that affects the way it prioritizes data, where it ignores the lower-variance components which is something that PCA already does. Therefore, lightGBM doesn't experience performance loss in this aspect. [7]

5 Conclusion

Through the application of different ML techniques, feature engineering and data processing, the goal was to find the prediction model that maximizes the AUC score (area under the ROC curve), which corresponds to its classification performance of activity values. To summarize, the project include two different choice of representing the dataset:

- Full feature representation: Evaluating the prediction values for the 'ACTIVE' label while considering extracted features and dropping the less informative ones: 'INDEX' and 'SMILES' column.
- Dimensionality reduced representation: Applying PCA for the fingerprint features, while preserving the original descriptor molecules.

Two machine learnings of choice for development and comparison included

- Random Forest
- LightGBM

Random Forest achieved the highest AUC score of 0.91, outperforming LightGBM and the PCA-reduced representation for the fingerprints. This also concludes that fingerprints disclose important information about the essential molecular structures and substructures, which in turn affects the models performance. Therefore, result also highlights the strength of Random Forest when applied to the full feature set without dimensionality reduction. Besides this, the experimentation of this assignment uncovered many key insights, especially the importance of making sure to preserve the critical information during dimensional reduction.

6 References

References

[1] The Pecan Team, "Data Preparation for Machine Learning: A Complete Guide," Pecan AI Blog, Nov. 21, 2023. [Online]. Available: `https://www.pecan.ai/blog/data-preparation-for-machine-learning/`

[2] Awan, Ali A., "The Curse of Dimensionality in Machine Learning: Challenges, Impacts, and Solutions,'" *DataCamp Blog*, Sep.

13, 2023. [Online]. Available: https://www.datacamp.com/blog/curse-of-dimensionality-machine-learning. [Accessed: 23-Dec-2024].

[3] Hossain. Mohtasim, M. "Mastering LightGBM: An In-Depth Guide to Efficient Gradient Boosting," Medium. Jan 8, 2024. [Online]. Available: https://shorturl.at/o8OPw. [Accessed: 23-Dec-2024].

[4] LightGBM, "LightGBM Parameter Tuning," LightGBM Documentation. [Online]. Available: https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html. [Accessed: 23-Dec-2024].

[5] LightGBM, "LGBMClassifier — LightGBM 4.1.0 documentation," LightGBM Documentation. [Online]. Available: https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html. [Accessed: 23-Dec-2024].

[6] Scikit-learn, "sklearn.ensemble.RandomForestClassifier," Scikit-learn Documentation, version 1.5. [Online]. Available: https://scikit-learn.org/1.5/modules/generated/sklearn.ensemble.RandomForestClassifier.html. [Accessed: 23-Dec-2024].

[7] GeeksforGeeks, "LightGBM Boosting Algorithms," *GeeksforGeeks*, Oct. 2023. [Online]. Available: https://www.geeksforgeeks.org/lightgbm-boosting-algorithms/

[8] GeeksforGeeks, "Check if a given matrix is sparse or not," *GeeksforGeeks*, [Online]. Available: https://www.geeksforgeeks.org/check-given-matrix-sparse-not/. [Updated: 20 juli 2022]. [Accessed: Dec. 23, 2024].

[9] FasterCapital, "Learning Rate: Slow and Steady - The Impact of Learning Rate on Gradient Boosting," 2024. [Online]. Available: https://fastercapital.com/content/Learning-Rate--Slow-and-Steady--The-Impact-of-Learning-Rate-on-Gradient-Boosting.html. [Accessed: Dec. 24, 2024].