VISMAI - Drug discovery virtual screening tool

1. Introduction

Drug discovery is a complex and costly process often marked by high failure rates and lengthy development times. Recent advancements in computational methods and machine learning have the potential to accelerate and refine this process. This study presents VISMAI, a comprehensive virtual screening tool that integrates modules for molecular descriptor calculation, fingerprint generation, and drug-likeness prediction. The descriptor calculation module assesses chemical properties from molecular structures, while the fingerprint generation module creates molecular fingerprints from SMILES strings. The drug-likeness prediction module utilizes advanced graph-based neural networks, tested alongside other graph-based and machine learning models, to enhance accuracy. VISMAI aims to improve predictive performance and efficiency in drug-likeness evaluation, with future efforts directed towards refining model interpretability and expanding datasets. This work highlights the significant impact of computational approaches on advancing pharmaceutical research and fostering innovation.

2. Application Overview

The virtual screening tool, VISMAI, is a web-based application that facilitates the input of chemical compounds via SMILES (Simplified Molecular Input Line Entry System) notations. This tool computes an extensive array of physicochemical properties, drug-likeness metrics, absorption and distribution characteristics, and potential toxicity risks. By leveraging these computational insights, VISMAI aids researchers in efficiently identifying promising drug candidates.

The application comprises three primary modules:

1. Molecular Descriptor Calculator
2. Fingerprint Calculator
3. Drug-Likeness Prediction

The application interface provides foundational information on drug discovery, details about the modules, an input and evaluation section, as well as contact and additional details.

Compound Name	Chemical Formula	Canonical SMILES
Amiloxate	C15H20O3	COc1ccc(=C(=O)OCCC(C)C)cc1
Ibuprofen	C13H18O2	CC(C)Cc1ccc(cc1)C(C)C(O)=O
Minoxidil	C9H15N5O	NC1=CC(=NC(=N)N1O)N2CCCCC2

Table 4.1: Example compounds along with their chemical formula and corresponding SMILES notations

2.1 Molecular Descriptor Calculator

This module calculates 24 molecular descriptors, including ADMET properties, using RDKit based on user-provided SMILES notations. The results are available for download in CSV format for subsequent evaluation.

```
#Sample list of molecular descriptors using RDkit
result.get('smiles', ''),
result.get('molecular_formula', ''),
result.get('molecular_weight', ''),
result.get('logP', ''),
result.get('tpsa', ''),
result.get('num_rotatable_bonds', ''),
result.get('num_h_acceptors', ''),
result.get('num_h_donors', ''),
result.get('num_heavy_atoms', ''),
result.get('molar_refractivity', ''),
result.get('fraction_csp3', ''),
result.get('logD', ''),
result.get('logS', ''),
result.get('bioavailability_score', ''),
result.get('num_rings', ''),
result.get('num_hetero_atoms', ''),
result.get('formal_charge', ''),
```

```
result.get('num_rigid_bonds', ''),
result.get('polar_surface_area', ''),
result.get('sp3_count', ''),
result.get('radical_electrons', ''),
result.get('valence_electrons', ''),
result.get('heavy_atom_molecular_weight', ''),
result.get('exact_molecular_weight', ''),
result.get('qed', ''),
result.get('ab_p_glycoprotein', ''),
result.get('ab_human_intestinal_absorption', ''),
result.get('ab_protein_binding_percentage', ''),
result.get('ds_blood_brain_barrier', ''),
result.get('ds_fraction_unbound', ''),
result.get('li_alogp', ''),
result.get('li_xlogp', ''),
result.get('li_ilogp', ''),
result.get('li_wlogp', ''),
result.get('metabolism', ''),
result.get('ex_clearance', ''),
result.get('ex_intrinsic_clearance', ''),
result.get('ex_half_life', ''),
result.get('tx_high_logP', ''),
result.get('tx_toxicity_score', ''),
result.get('dl_lipinski_rule', ''),
#result.get('dl_lipinski_conditions', ''),
result.get('dl_veber_rule', ''),
result.get('dl_ghose_rule', ''),
result.get('dl_egan_rule', ''),
result.get('dl_pfizer_rule', ''),
])
```

2.2 Fingerprint Calculator

The Fingerprint Calculator module generates fingerprints for multiple SMILES and allows for their download in CSV format. The tool can produce:

- 1024-bit Morgan fingerprints
- 2048-bit Morgan fingerprints
- 166-bit MACCS fingerprints
- 2048-bit Torsion fingerprints
- 2048-bit RDK fingerprints
- 16383-bit Atom Pair fingerprints
- 1024-bit Avalon fingerprints

2.3 Drug-Likeness Prediction

This module employs the XGBoost algorithm to assess whether a compound is drug-like or non-drug-like. It supports the evaluation of multiple SMILES simultaneously.

Our model for drug-likeness prediction utilizes several key components, each designed to handle different aspects of graph-structured data and enhance prediction accuracy. Below is a detailed description of each component, including code snippets that illustrate their implementation.

2.3.1 Understanding Graph Convolutional Neural Networks (GCNNs)

Graph Convolutional Neural Networks (GCNNs) are a specialized type of neural network designed to operate on graph-structured data. They extend the concept of traditional convolutional neural networks (CNNs), which are typically used for grid-like data (such as images), to handle the complexities of graphs. Here's a breakdown of their key features and functionalities:

1. Graph Structure

- **Nodes and Edges**: In a graph, data is represented as nodes (vertices) connected by edges (links). Each node can have associated features, making it suitable for various applications, including social networks, molecular structures, and more.

2. Convolution on Graphs

- **Local Neighborhood**: GCNNs perform convolution operations by aggregating information from a node's local neighborhood. This allows the network to learn from the relationships and interactions between connected nodes.
- **Feature Extraction**: By propagating features from neighboring nodes, GCNNs can extract meaningful patterns and representations from the graph data.

3. Key Components

- **Graph Convolution Layers**: These layers are the building blocks of GCNNs, where convolution operations are performed directly on the graph structure. They leverage the adjacency matrix of the graph to facilitate information flow between nodes.
- **Pooling Layers**: Similar to traditional CNNs, pooling layers in GCNNs help reduce dimensionality and capture essential features by summarizing information from neighboring nodes.

Our GCNN implementation

1. Graph Convolutional Layers

The core of our model is built around Graph Convolutional Layers. These layers perform convolution operations on molecular graphs, where atoms are nodes and bonds are edges. The Graph Convolutional Layers learn node-level features that capture the local structure of each molecule.

```
class _GraphConvKerasModel(tf.keras.Model):
    def __init__(self, graph_conv_layers, **kwargs):
        super(_GraphConvKerasModel, self).__init__(**kwargs)
        self.graph_convs = [
            layers.GraphConv(layer_size, activation_fn=tf.nn.rel
            for layer_size in graph_conv_layers
        ]
```

Key Features:

- **Message Passing:** Each node aggregates information from its neighbors, which helps the model learn from the local chemical environment around each atom.
- **Layer-wise Aggregation:** Multiple layers of graph convolution capture complex relationships and patterns in the molecular structure

2. Graph Pooling Layers

After applying Graph Convolutional Layers, we use Graph Pooling Layers to aggregate node-level features into a fixed-size graph-level representation. This pooling operation consolidates information from all nodes in a molecule into a single vector.

```
self.graph_pools = [layers.GraphPool() for _ in graph_conv_layer
```

Key Features:

- **Dimensionality Reduction:** Graph Pooling reduces the dimensionality of the graph representation, simplifying processing.
- **Feature Aggregation:** Combines information from various nodes to form a comprehensive feature vector representing the entire molecule.

3. GraphGather Layer

The GraphGather Layer consolidates the fixed-size outputs from the Graph Pooling Layer into a final graph-level representation. This layer ensures that the model output aligns with the number of samples in the batch.

```
self.graph_gather = layers.GraphGather(batch_size=batch_size, ac
```

Key Features:

- **Fixed-Size Output:** GraphGather produces a fixed-size output vector for each molecule, regardless of the number of nodes.

- **Sample Alignment:** Ensures that the output tensor matches the number of molecules in the batch, essential for subsequent classification or regression tasks

4. Dense Layers

Following the aggregation of node-level features, Dense Layers further process the graph-level representations. These fully connected layers help in learning complex patterns from the aggregated features.

```
self.dense = Dense(dense_layer_size, activation=tf.nn.relu)
self.dense2 = layers.Dense(256)
```

Key Features:

- **Non-Linearity:** Dense Layers introduce non-linearity into the model, enhancing its ability to capture intricate relationships in the data.
- **Dimensionality Expansion:** Multiple Dense Layers with varying sizes refine the feature representations before making final predictions.

5. Output Layers

For classification tasks, the final output is a vector of class probabilities for each molecule. This is achieved through:

Binary Classification:

```
self.reshape_dense = Dense(n_tasks * n_classes)
self.reshape = Reshape((n_tasks, n_classes))
self.softmax = Softmax()
```

- **Softmax Layer:** Converts the final dense layer output into probabilities, indicating the likelihood of each molecule being drug-like or non-drug-like.
- **Output Shape:** Given a batch of 32 molecules, the output is a 32×1 vector where each entry represents the probability of the corresponding molecule being drug-like.

Enhancements Over Standard GCNN

To improve performance and adapt to the specific requirements of drug-likeness prediction, we incorporated several enhancements over the standard GCNN:

Batch Normalization:

```
self.batch_norms = [
    BatchNormalization(fused=False) if batch_normalize else None
    for _ in range(len(graph_conv_layers) + 1)
]
```

Applied to stabilize and accelerate training by normalizing the inputs to each layer.

Dropout:

```
self.dropouts = [
    Dropout(rate=rate) if rate > 0.0 else None for rate in dropo
]
```

Added to prevent overfitting by randomly dropping units during training.

Our Graph Convolutional Neural Network model effectively captures the complex relationships in molecular graphs to predict drug-likeness. By integrating advanced layers such as Graph Pooling, GraphGather, and specialized Dense Layers, we have enhanced the standard GCNN architecture to address the specific need of molecular classification.

Conclusion

The development of the VISMAI virtual screening tool marks a significant advancement in streamlining the drug discovery process. Utilizing advanced machine learning techniques, notably Graph Convolutional Neural Networks (GCNNs), VISMAI enhances drug-likeness prediction—a key factor in early drug

development. Integration of RDKit and Mordred libraries facilitates robust calculation of molecular descriptors and fingerprint generation, offering valuable insights into the chemical properties and structural characteristics of potential drug candidates.

Experimental results indicate that the GraphConv model surpasses the SAGEConv model in accuracy, precision, recall, and F1 score. Nevertheless, further refinements, such as hyperparameter optimization and exploration of alternative GCNN architectures, are needed to enhance predictive performance.

Future efforts will focus on improving model interpretability to provide clearer insights into the factors driving drug-likeness predictions. Expanding the dataset and incorporating diverse chemical spaces will further refine the tool’s predictive capabilities, ensuring its long-term relevance and effectiveness in drug discovery.