Helping chemists design selective JAK inhibitors

sainath reddy bobbala.
*sainathreddyb@usf.edu

***Abstract*—In this paper, I have evaluated different machine learning approaches for predicting protein-ligand binding, specifically the binding affinity measured by pKi and pIC50. Protein-ligand binding is a key step in drug discovery, and understanding the molecular interactions between proteins and small molecules is crucial for the development of selective and effective drugs.**

I will investigate various machine learning models and techniques for predicting the binding affinity of small molecules to the JAK family of proteins. Our objective is to find an approach that can accurately predict the values of pKi and pIC50 and that can be used as a powerful tool for virtual screening in drug discovery.

Furthermore, I will evaluate the performance of different models, including, but not limited to random forest, gradient boosting, transformers, graph networks, and I will also investigate the impact of various features such as molecular descriptor, fingerprint, and graph convolutional networks on the prediction performance.

I. Introduction

Proteins play a crucial role in the functioning of the human body, and many diseases arise from proteins being either overactive or inactive. Inhibitors, which stop the activity of proteins, are a common type of drug used to treat these diseases. Medicinal chemists aim to design molecules that can fit perfectly into a protein's "keyhole", similar to a key opening a lock. However, many proteins have similar keyholes, leading to drugs that can affect multiple unintended proteins and cause side effects. The goal is to create molecules that are selective and only interact with the targeted protein.

Machine learning has the potential to aid in this process by predicting the ability of molecules to stop various proteins. In this research, the focus will be on the JAK (Janus-associated kinase) family of proteins, which are strongly linked to diseases such as cancer and inflammation. There are currently several JAK inhibitors on the market to treat human diseases, but they are often unselective and may affect other proteins in the JAK family. The pharmaceutical industry has thus made it a priority to design selective JAK inhibitors with fewer side effects.

II. Exploratory Data Analysis

EDA is better shown in Jupyter notebooks, please, look at the zip file for the EDA notebook.

III. Input Processing

In this section, I will describe various things involved with the given data before training the models.

A. Data splitting

Stratification of data is important because it ensures that the different subgroups within a population are represented fairly in the sample. This is particularly important in situations where population subgroups have different characteristics or responses to a phenomenon of interest.

I have used 80, 10, 10 rations for the train, test and val sets, respectively, and these splits are made so that a compound present in train test should not be present in the test, val set and vice versa. The reason behind doing this is, for example, let $c1$ be a compound, and it has values of pKi against all 4 kinases divided into 2 sets, if the first set is present in the train and the second set in the test, then the model will not be able to fit properly as it does not have a clear picture of all 4 kinases with respect to this compound, and at test time it may just give the value near the values in the train set pKi values irreparable of the actual label. Once the splits are made, I have manually verified if the splits maintain a balanced ration of each kinase type and measurement type across different splits.

B. Label Scaling

I decided to use min-max scaling and transformed the label values (pKI/pIC50 values) into a range of 0 to 1. This was done because the original values fell within a specific range (6 to 12) and there were no significant outliers. Additionally, values whose z-score was greater than 3 were considered important molecules with high inhibition values and should not be considered outliers. Furthermore, it is beneficial to scale the values to between 0 and 1 because input features, such as molecular fingerprints, are also in the same range. Therefore, it is optimal to scale the inputs to match the range of the outputs.

I attempted to see the performance by using separate scalers for each measurement type to scale the output labels, however, this resulted in a slight reduction in performance. This method may still be useful in the future if new measurement types with different ranges are introduced, as using a separate scaler for each type would be more appropriate in that case.

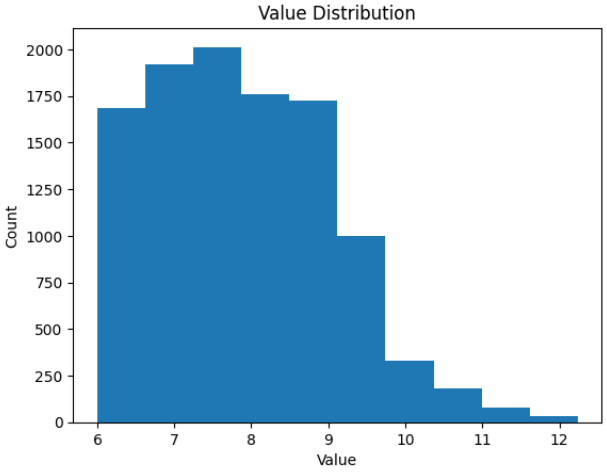
C. Kinase type and measurement type Feature encoding

Kinase type and measurement type are used as input to the model, as both features are categorical and encoded into one hot vector, and this represents a vector of size 6.

D. Feature Engineering

Molecular fingerprints, also known as molecular descriptors, are numerical representations of the chemical structure of a molecule. They are used to describe the molecular structure

Fig. 1. distribution of pKI/pIC50 values.



and properties in a way that can be easily compared and analyzed by computational methods.

1) Morgan Fingerprint: Morgan fingerprint, also known as circular fingerprint, is a type of molecular fingerprint that is based on the topology of a molecule. The Morgan fingerprint is computed by creating a circular variant of the atom pair fingerprint (APFP). The basic idea behind this approach is to represent a molecule as a graph where atoms are the vertices, and bonds are the edges. Then a circular path (or "circular fingerprint") is created by starting from a specific atom, and moving along the bonds to other atoms, until returning to the starting atom. This process is repeated for all atoms in the molecule, and the resulting circular paths are hashed to a fixed-length vector, which forms the Morgan fingerprint.

2) RDKFingerprint: The RDKFingerprint is computed by creating a circular path (or "circular fingerprint") similar to Morgan fingerprints, but it also allows for the use of different radius values. A radius of 1 will only consider the immediate neighbors of an atom, while a larger radius will include more distant atoms. Additionally, the RDKFingerprint also allows the use of different hashing functions, and it can also include additional features, such as atom types and bond types.

3) MACCS: MACCS (Molecular ACCess System) keys are a type of molecular fingerprint that is based on the chemical characteristics of a molecule. It is a set of 166 predefined structural keys, or features, that represent the most common functional groups and substructures found in small molecules.

All the molecular fingerprint values are calculated for each smile string in the dataset given and highly correlated columns from fingerprint are removed.

It is common practice to remove highly correlated columns before training a model, as highly correlated columns can cause problems in the model training. This is because highly correlated columns provide almost the same information, which can lead to overfitting, or cause the model to give more weight to one of the correlated columns, even though it is not the most informative.

To identify highly correlated columns, I have computed the correlation matrix of the input variables and then removed the columns that have a correlation coefficient greater than a certain threshold of 0.9. The overall process can be described mathematically as follows:

1) SMILES string $\rightarrow$ Molecular Object
2) Molecular Object $\rightarrow$ Fingerprints (F_1, F_2, F_3)
3) Correlation matrix = corr(F_1, F_2, F_3)
4) Highly correlated columns = columns where corr$(F_1, F_2, F_3) >$ threshold
5) Fingerprints $(F_1, F_2, F_3) \rightarrow (F_1 - C_1, F_2 - C_2, F_3 - C_3)$

TABLE I
PERFORMANCE OF RANDOM FOREST WITH MOLECULE FINGERPRINT AS INPUT.

model name	val MAE	test MAE
rf + mol fingerprint	0.065	0.064

E. feature processing for graph networks

Graph Neural Networks (GNNs) take the whole graph as input, which is represented as an adjacency matrix or an adjacency list, and features associated with each node or edge in the graph, I followed the approach shown in [1] where on top of converting the graph into adjacency matrix they have extracted various node features such as:

1) Atomic number: number of protons in the nucleus of that atom
2) chirality : property of certain molecules that are not superimposable in their mirror images.
3) bond type : whether the bond is dingle, double, aromatic etc.

There are in total 9 features per each atom, and these are internally collected using rdkit library. Along with these features, conformer features are also collected

IV. METHODOLOGY

A. Random Forest(rf)

Considering the sample size, I have started with simple random forest and linear regression models and used the molecular fingerprint, kinase type, and measurement type features calculated as shown above as input and predicted the measurement value. And table 1 IV-A show the MAE on test and val set. As the random forest used the bagging approach and builds weak learners by selecting a subset of features in each split, this helps to reduce overfitting. And as linear regression expects input features to have linear relation to output labels, random forests do not make any such assumptions and can fit complex nonlinear relations that exist between input and output.

Performance of the model by adding conformer features need to be analyzed further, so not including this in this version.

TABLE II
PERFORMANCE OF COATGIN MODEL.

model name	val MAE	test MAE
gnn	0.0864	0.082
gnn + mol fingerprint	0.074	0.078
gnn + mol fingerprint + conformer	na	na

TABLE III
PERFORMANCE OF THE TRANSFORMER MODEL.

model name	Val MAE	Test MAE
transformer without training	0.1097	0.1083
transformer training	0.12	0.12
transformer training+ mol fingerprint	0.0967	0.0953
transformer without training+ mol fingerprint	0.067	0.062

B. Graph Neural Networks

I have used the CoatGIN architecture [2] considering its success in predicting molecular properties. CoAtGIN is a graph neural network architecture that has a few parameters, only 5.2 million, making it easy to train on the GPU I have access to (12 GB of memory) and computationally cheaper to train. The architecture utilizes various skip connections, which act as regularization by allowing gradients to flow more easily through the network and by allowing the network to learn more robust and generalizable representations. Due to its small number of parameters and considering the size of the given dataset, I have used this model. And table IV-B show the performance of this model. There is a slight improvement in MAE but not a big push, I will discuss briefly about these results further in evaluation section.

C. Transformer

I have used Roberta-base transformer model from hugging face which is pretrained using masked language modelling with byte level tokenizer. And the architecture is as shown in the figure 2. Due to its large size of training and small dataset, the transformer end-to-end resulted in less performance; this will be further discussed in the Evaluation section.

D. Training

All the models were trained on a 12 GB 1080Ti GPU with a batch size of 200. The predictions on the test and validation sets were made using the best model, which was selected based on the lowest MAE value achieved over 500 training epochs. Regularization was done using dropout layers in both models. As I have modeled it as a regression problem, I have chosen to minimize L1 loss for both models.

V. EVALUATION

This table V shows the results of three different models (rf, Transformer, and CoATGIN) evaluated on four different metrics: val MAE, test MAE, val+test+unscaled MAE, and val+test+pKi accuracy and val+test+plc50 accuracy.

1) Mean Absolute Error (MAE) → This is a commonly used evaluation metric for regression tasks, where the average difference between the predicted values and the actual values is calculated. The formula for MAE is:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Fig. 2. transformer model architecture.

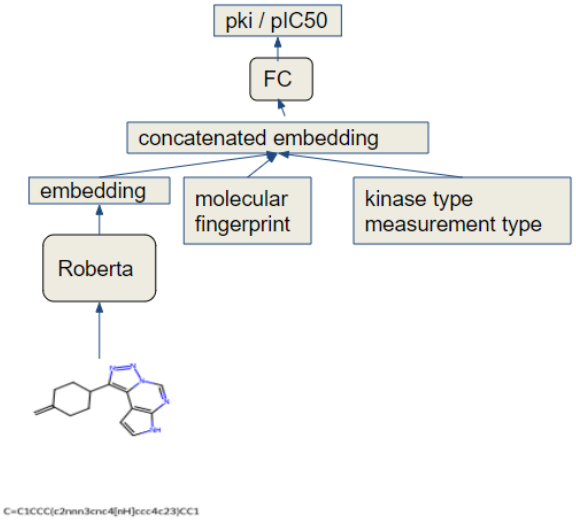


TABLE V
RESULTS ON DIFFERENT METRICS

metric	rf	Transformer	CoATGIN
val mae	**0.065**	0.067	0.074
test mae	0.064	**0.062**	0.078
val+test+unscaled mae	**0.39**	**0.39**	0.45
val+test+pKi accuracy	**0.78**	0.744	0.734
val+test+plc50 accuracy	**0.81**	**0.81**	0.78

2) Un scaled MAE → This is the MAE in actual label values i.e in range 6 to 12 between predictions and labels
3) val+test+pKi/ plc50 accuracy → This is a ratio of accurate predictions divided by total unique SMILES strings in that measurement type. A compound C is considered accurate if it has a high pKi/plc50 value for JAK1 and the model gives a high value for JAK1. The formula for val+test+pKi/plc50 accuracy is:

$$val + test + pKi/plc50 \quad accuracy = \frac{number\ of\ accurate\ predictions}{total\ unique\ SMILES\ strings\ in\ that\ measurement\ type}$$

This metric is designed to reflect the actual use case of finding the best inhibitor, which has high inhibition for one compound and low inhibition for others.

On the val MAE metric, the rf model has the best performance with a score of 0.065. On the MAE test metric,

the Transformer model has the best performance with a score of 0.062. On the val+test+unscaled MAE and val+test+pKi accuracy and val+test+plc50 accuracy metrics, both the rf and Transformer models have the best performance with a score of 0.39 and 0.81, respectively. In general, the rf model performs best on the validation set, while the transformer model performs best on the test set. However, the difference in performance between the models is not very significant.

In addition to the commonly used evaluation metrics, other metrics could be introduced to provide a more comprehensive analysis of the model's performance. These metrics could include the ordering of the predictions for each kinase, as well as ratios of the inhibition values between different kinases (e.g. JAK1/JAK2 ratio). These metrics would provide a clearer picture of the model's performance and help to identify any potential issues or areas for improvement.

A. The Overfitting Problem

In this study, I have faced the challenge of overfitting when using graph and transformer-based models. Specifically, training end-to-end with transformers has led to decreased performance compared to freezing the transformer layers. Despite implementing regularization techniques such as dropout, I observed that the training loss continued to decrease, while the validation loss remained high.

This can be attributed to the high complexity of the transformer-based models, which leads to overfitting by memorizing the training data. To mitigate this problem, we have employed the strategy of freezing some layers of the transformer-based models during training.

So as shown in table IV-C transformer model performed very well when the layers are frozen. $Val\ Loss = Bias^2 + (1-f) \times Variance + Irreducible\ Error$ So according to above equation f refers to the number of layers frozen, so we can reduce variance by freezing the layers, and keeping bias in check.

In contrast to transformers and GNN, this was not observed in random forest. Firstly, Random Forest uses the ensemble method of bagging and subsampling, where multiple decision trees are constructed, and their predictions are combined to improve the overall performance.

Secondly, correlated features were removed from the dataset. Correlated features tend to introduce redundant information and can lead to overfitting in machine learning models. By removing correlated features, I have reduced the complexity of the model and improved its generalization ability.

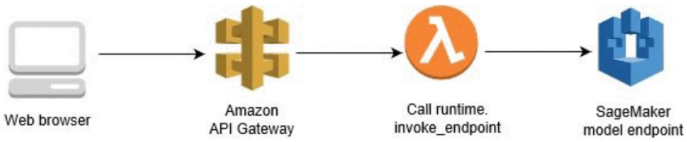
The combination of the above two techniques has resulted in a robust Random Forest model that can effectively handle high-dimensional and correlated data while avoiding overfitting.

B. The Data Imbalance Problem

Our analysis of the data revealed that the distribution of kinases is relatively similar and we have a limited number of samples for the PKI measurement type. This has led to a lower performance for PKI compared to pIC50 in our model.

The challenges identified in this research will be further investigated in the future improvement section, where possible solutions to overcome them will also be proposed.

Fig. 3. Simple Model deployment.



VI. Model Deployment

AWS offers several options for deploying models, including EC2, EKS, ECS, and Sage Maker. Among these, Sage Maker is the easiest to set up. Sage Maker is a fully managed machine learning service that supports both model inference and training. As depicted in the diagram 3, we can use Sage Maker to deploy a serialized model by storing its artifacts in an S3 bucket, using the code provided below. And this models sits behind a serverless lambda endpoint which gets invoked by an API gateway service.

Code Listing 1. Pseudo code for model deployment in sage maker

```
import boto3
import sagemaker

# create a SageMaker session
sagemaker_session = sagemaker.Session()

# upload the model to an S3 bucket
model_path = sagemaker_session.
upload_data(path='model.pth', key_prefix='model')

# create a SageMaker model
model = sagemaker.pytorch.model.
PyTorchModel(model_data=model_path,
     role=sagemaker.get_execution_role(),
     framework_version='1.4.0',
     entry_point='predict.py',
     source_dir='code')

# deploy the model to an endpoint
predictor = model.deploy(initial_instance_count=1,
instance_type='ml.m4.xlarge')
```

Code Listing 2. lambda handler for sagemaker endpoint

```
import boto3
import json

# define the Lambda function
def lambda_handler(event, context):
    # extract the input data from the event
    input_data = json.loads(event['body'])

    # use the SageMaker endpoint to make a prediction
    response = predictor.predict(input_data)

    # return the response in the desired format
    return {
        'statusCode': 200,
        'body': json.dumps(response)
    }
```

From the Front end if it's in python we can use Boto package to invoke the API gateway endpoint. But for each language, AWS has its own clients to call its service after authenticated.

Code Listing 3. post to call to aws api gateway to get predictions

```
import boto3
import json

client = boto3.client('apigateway')

def invoke_api_gateway(api_id, resource_id, method, body):
    response = client.test_invoke_method(
        restApiId=api_id,
        resourceId=resource_id,
        httpMethod=method,
        body=json.dumps(body)
    )
    return json.loads(response['body'])

# Example usage
api_id = 'your_api_id'
resource_id = 'your_resource_id'
method = 'POST'
body = {'smiles': 'CC(=O)Oc1ccccc1C(=O)O', 'measurement_type':
'pIC50', 'kinase_name': 'JAK1'}
response = invoke_api_gateway(api_id, resource_id,
method, body)
print(response)
```

VII. Future Improvements

In this section, I will write about future ideas that I want to work on given time and resources.

A. Self-Supervised learning for pre-training

To enhance future performance, one approach is to utilize self-supervised learning techniques for pretraining models, as this approach has been demonstrated to be effective in past studies. My own research found that using a pretrained Roberta model with masked language modeling resulted in improved performance when compared to using Graph Neural Networks without any pretraining. To further improve performance, incorporating additional labeled data from related datasets such as MolNet, Zinc, Tox21 can be considered. In addition, utilizing contrastive learning methods, such as the student-teacher setting or training two models together to align the latent representations, such as a Graph Network that works well on well-defined structures and a transformer that is efficient in handling long term dependencies can be used. we can also use graph features such as pair wise distance between atoms given a 3d structure as an initial labels and do supervised pretraining step and later this can be fine-tuned for actual property prediction. Along with these data augmentation on smile structure can be studied which helps in reducing overfitting

B. More Chemical Features

By just adding molecular fingerprints performance has improved, I have prepared conformer features as well but need to add them to the models and studied. So more chemical features can be extracted from rd kit toolkit at node level. Multi Modal learning can be tried by combining different datasets.

C. Modelling as a classification Problem

Small range regression problems can be converted into classification by dividing the continuous range of the target variable into discrete intervals. This approach can be useful when there are clear boundaries between different levels or states and the goal is to make a binary or multi-class decision. And given our problem of finding the chemical compound that inhibits the kinase, we can consider this as 4 way classification such highly inhibited, mildly inhibited, less mildly inhibited, very less inhibition.

VIII. Conclusion

In conclusion, there are several ways to improve the performance of models for predicting properties of chemical compounds. One approach is to utilize self-supervised learning techniques for pretraining models, incorporating additional labeled data from related datasets, and using contrastive learning methods. Additionally, incorporating more chemical features, such as conformer features and features extracted from the RDKit toolkit, can improve performance.

Given more time I want to further understand the nature of failures in the predictions.

References

[1] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec, "Ogb-lsc: A large-scale challenge for machine learning on graphs," 2021. [Online]. Available: https://arxiv.org/abs/2103.09430

[2] X. Zhang, C. Chen, Z. Meng, Z. Yang, H. Jiang, and X. Cui, "Coatgin: Marrying convolution and attention for graph-based molecule property prediction," *bioRxiv*, 2022. [Online]. Available: https://www.biorxiv.org/content/early/2022/08/29/2022.08.26.505499