To Kill a Mycobacterium:

Novel Antibiotic Discovery in *Mycobacterium tuberculosis H37Rv*

Marena Trinidad

INTRODUCTION

One-third of Earth's population is a carrier of M. tuberculosis, culminating in 2 million deaths a year[1]. However, humanity faces an even greater challenge, for within The US alone, 10,042[2] patients develop treatment-resistant tuberculosis, with infections impervious to all clinically-available antibiotics. This poses a major threat to public health, as natural selection further increases the global presence of antibiotic-resistant tuberculosis. Subsequent effects could be catastrophic and citizen scientists have aligned with The WHO and other organizations to fulfill the gaping demand for novel antibiotics.

This experiment is one such attempt, geared against one of the most notorious strains of M. tuberculosis, H37Rv. Hereby, publicly available datasets were curated from high-throughput, H37Rv chemical-screens and scrutinized in hopes of developing a pharmacophore-driven model capable of predicting bactericidal activity in uncharacterized compounds.

METHODS

DATA COLLECTION

For the purposes of this study, PubChem BioAssay AID1332[4] was web-scraped to substantiate a training dataset. The amalgamated information comprises a suite of 10,466 compounds, all of which were screened *in vitro* with H37Rv. Compound activity was classified by observing chemical lethality over a range of concentrations and timepoints. Chemicals that incurred >90% fatality amongst its initial bacterial-population were labeled "Active", whereas the remaining compounds were annotated as "Inactive." All activity assays were executed robotically in a highly controlled experiment as previously described [4].

For the drug-discovery purposes of this experiment, a natural-product database was stockpiled through granted access to repositories from The International Bio Screens Ltd.[6] and The Collaborative Drug Discovery Vault[3] . The assembled dataset contained itemized string representations of 64,566 compounds distilled from naturally occurring sources, varying from plant, marine and microbiological communities. Due to intellectual property constraints, this data is not provided in the project repository.

MOL-TO-VEC FEATURE REPRESENTATION

Each activity screen was annotated with molecular-structure data encoded as SMILES strings--a streamlined system for representing chemicals in a single line of text. Each compound was then vectorized using the cheminformatic software RDKit[5] (*Release_2016.03.1*), according to the Morgan Fingerprint convention, in a parsimonious computation leveraging binomial trees and hash-function compression. In this fashion, all molecules were divided into small fragments, and each unique fragment was assigned a label, whereby all molecules can be wholly represented as a binary array detailing its fragment constituents. This form of molecular representation is highly fidelic, capable of resolving chemical characteristics down to their enantiomeric and chiral levels. Extensive details about the vectorization are made available by the software[5] providers, but the principle of Circular Fingerprint generation is highly tried and displayed in ***Figure 1.***

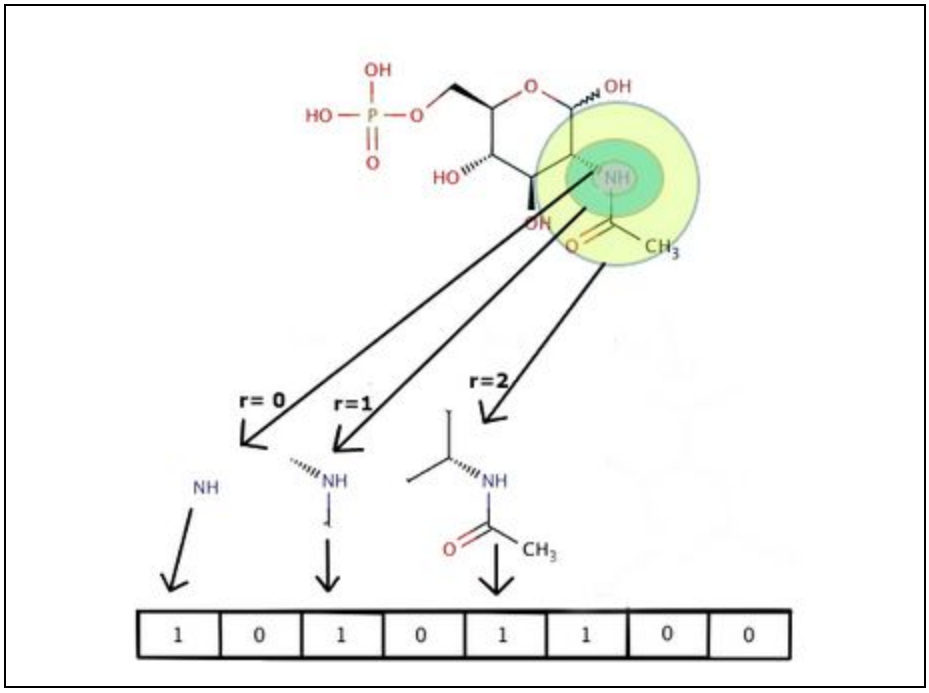


FIGURE 1: A simplified graphic detailing the Morgan Fingerprint process, in which a molecule is tokenized into fragments. Here, fragments are defined as any novel sequence of connected atoms, over a radius of 2, with the highlighted Nitrogen serving as a reference point. Unique fragment motifs are mapped to an array where each index serves as a binary feature detailing the inclusion of a fragment within the corporal lexicon.

In detail, circular fingerprinting interprets molecules as a network-connectivity graphs, where each atom is a node and each bond a directed edge. “Anchors” are set at each non-hydrogen atom and unique fragments are defined as all distinct connectivity sequences within an arbitrary radius of the set atomic reference-point. ***Figure 1*** depicts unique fragments designated within a molecule, rooted at a central, highlighted Nitrogen over a given radius of 2 bonds. The process continues until all fragments within the molecule are defined and incorporated into an optimized binary tree that details the complete fragment-space across an entire molecular corpus with expedited retrieval. A hash function compresses the tree into a 2048-bit vector space, such that every molecule within the corpus can be described by a binary array of length 2^{11}, with each index corresponding to a unique atomic-motif codified as a binary feature.

Code Block 1 contains the code for molecule-to-bit-vector computations. In custom with good practices and to manage computational complexity, fingerprint radii were set to a 2-bond maximum. The same fingerprinting parameters were applied to all datasets in the study for consistency between the train and test-screen compounds. More explicit details about the vectorization work-up are available in the project repository, within the Ipython Notebook labeled “TrainingSpace,” inside the “Code” Folder.

CODE BLOCK 1: SMILES-Vectorization Process in RdKit

```
# DEFINE FX: FINGERPRINT GENERATOR
def make_fingerprints(smiles, radius = 2):
    from rdkit import Chem
    from rdkit import DataStructs
    from rdkit.Chem import AllChem

    mols, np_fps, bit_fps, arr = [], [], [], []

    for m in smiles:
        mol = Chem.MolFromSmiles(m)
        mols.append(mol)
        fp = AllChem.GetMorganFingerprintAsBitVect(mol, radius)
        bit_fps.append(fp)
        arr = np.zeros((1,))
        DataStructs.ConvertToNumpyArray(fp, arr)
        np_fps.append(arr)
    return mols, bit_fps, np_fps

# GENERATE FINGERPRINTS
train_mol, train_bit, train_fps = make_fingerprints(chem_lib.Smiles)
```

EXAMINATION OF THE MOLECULAR-FEATURE SPACE

Due to the complex nature of the chemical vectorization process, a comprehensive examination of the molecular-fingerprint landscape is essential in understanding the implications and validity of any model built upon it. Chief among this task is uncovering the distribution of compound activities within the training corpus. That is, 241 active compounds exist along with 10,225 inactive compounds(***Figure 2***). This drastic class imbalance will merit significant stringency in model recall during validation.

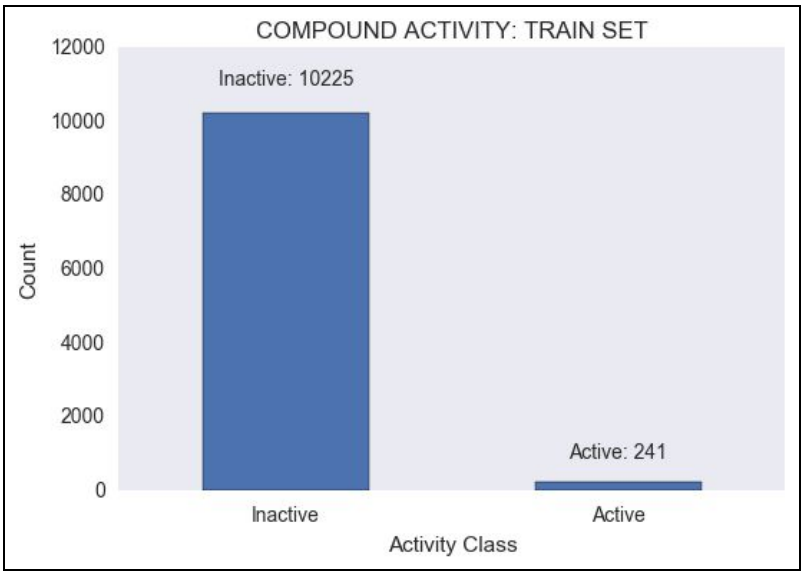


FIGURE 2: Bar chart demonstrating the activity-class distribution of the training set.

Deciphering each bit, and identifying its corresponding atomic-motif, offers a deeper insight into the feature space. Below, bit identities were unpacked (***Code Block 2***) and their respective frequencies were tabulated to isolate the most prevalent fragments within the corpus. Bit-specific frequencies are plotted in ***Figure 3***, along with summary statistics in ***Figure 4***, discovering fragment frequencies to vary widely (from 0 to 10,301), with the majority of fragments occurring less often than the median frequency and only a small proportion of bits existing at frequencies above the 75th percentile. To be specific, the corpus' most common fragments are depicted in ***Figure 5***. However, fragments frequencies across the corpus alone offer little help in discriminating between active and inactive compounds. Deeper analysis is required to quantify bit/feature importance and will be addressed within the Feature Engineering Section.

CODE BLOCK 2: Calculation of Bit Statistics and Visualizations

```
# CALCULATE BIT FREQUENCY STATS
from rdkit.Chem import AllChem as Chem
from rdkit import DataStructs
import pickle
from collections import Counter
%pylab inline

numBitCount = Counter()
fpBitCount = Counter()

for i in range(len(train_bit)):
    fp = train_bit[i]
    numBitCount[fp.GetNumOnBits()]+=1
    for bit in fp.GetOnBits():
        fpBitCount[bit]+=1
```

```
def includeRingMembership(s, n):
    r=';R]'
    d="]"
    return r.join([d.join(s.split(d)[:n]),d.join(s.split(d)[n:])])

def includeDegree(s, n, d):
    r=';D'+str(d)+']'
    d="]"
    return r.join([d.join(s.split(d)[:n]),d.join(s.split(d)[n:])])

def writePropsToSmiles(mol,smi,order):
    finalsmi = smi
    for i,a in enumerate(order):
        atom = mol.GetAtomWithIdx(a)
        if atom.IsInRing():
            finalsmi = includeRingMembership(finalsmi, i+1)
        finalsmi = includeDegree(finalsmi, i+1, atom.GetDegree())
    return finalsmi

def getSubstructSmi(mol,atomID,radius):
    if radius>0:
        env = Chem.FindAtomEnvironmentOfRadiusN(mol,radius,atomID)
        atomsToUse=[]
        for b in env:
            atomsToUse.append(mol.GetBondWithIdx(b).GetBeginAtomIdx())
            atomsToUse.append(mol.GetBondWithIdx(b).GetEndAtomIdx())
        atomsToUse = list(set(atomsToUse))
    else:
        atomsToUse = [atomID]
        env=None
    smi = Chem.MolFragmentToSmiles(mol,atomsToUse,bondsToUse=env,allHsExplicit=True, allBondsExplicit=True,
                                   rootedAtAtom=atomID)
    order = eval(mol.GetProp("_smilesAtomOutputOrder"))
    smi2 = writePropsToSmiles(mol,smi,order)
    return smi,smi2

def _prepareMol(mol,kekulize):
    from rdkit.Chem import rdDepictor

    mc = Chem.Mol(mol.ToBinary())
    if kekulize:
        try:
            Chem.Kekulize(mc)
        except:
            mc = Chem.Mol(mol.ToBinary())
    if not mc.GetNumConformers():
        rdDepictor.Compute2DCoords(mc)
    return mc
```

```
def moltosvg(mol,molSize=(450,200),kekulize=True,drawer=None,**kwargs):
    from IPython.display import SVG
    from rdkit.Chem.Draw import rdMolDraw2D

    mc = _prepareMol(mol,kekulize)
    if drawer is None:
        drawer = rdMolDraw2D.MolDraw2DSVG(molSize[0],molSize[1])
    drawer.DrawMolecule(mc,**kwargs)
    drawer.FinishDrawing()
    svg = drawer.GetDrawingText()
    return SVG(svg.replace('svg:',''))

def getSubstructDepiction(mol,atomID,radius,molSize=(450,200)):
    if radius>0:
        env = Chem.FindAtomEnvironmentOfRadiusN(mol,radius,atomID)
        atomsToUse=[]
        for b in env:
            atomsToUse.append(mol.GetBondWithIdx(b).GetBeginAtomIdx())
            atomsToUse.append(mol.GetBondWithIdx(b).GetEndAtomIdx())
        atomsToUse = list(set(atomsToUse))
    else:
        atomsToUse = [atomID]
        env=None
    return moltosvg(mol,molSize=molSize,highlightAtoms=atomsToUse,highlightAtomColors={atomID:(.1,.1,1)})

def depictBit(bitId,examples,mols,molSize=(450,200)):
    zid = examples[bitId]
    info={}
    fp = Chem.GetMorganFingerprintAsBitVect(mols[zid],2,2048,bitInfo=info)
    aid,rad = info[bitId][0]
    return getSubstructDepiction(mols[zid],aid,rad,molSize=molSize)
```

```
# CREATE IMAGE AND STATS DF FOR ALL BITS
import pandas as pd
from rdkit.Chem import PandasTools
PandasTools.RenderImagesInAllDataFrames(images=True)

rows_c = []
for bCount,bitId in sorted(itms,reverse=True):
    pccid = bitExamples[bitId]
    if pccid in keepMols:
        info={}
        fp = Chem.GetMorganFingerprintAsBitVect(keepMols[pccid],2,2048,bitInfo=info)
        aid,rad = info[bitId][0]
        smi1,smi2 = getSubstructSmi(keepMols[pccid],aid,rad)
        svg = depictBit(bitId,bitExamples,keepMols,molSize=(250,125))
        rows_c.append([bitId,
                       rad,pccid,svg.data,bCount])
train_bit_df = pd.DataFrame(rows_c,columns=('Bit_ID','Radius', 'PC_CID','drawing','counts'))
```

FIGURE 3: Bit Frequency

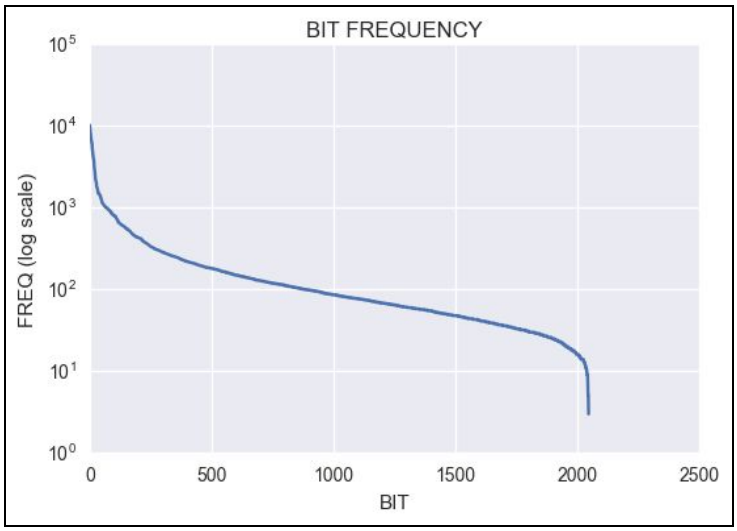
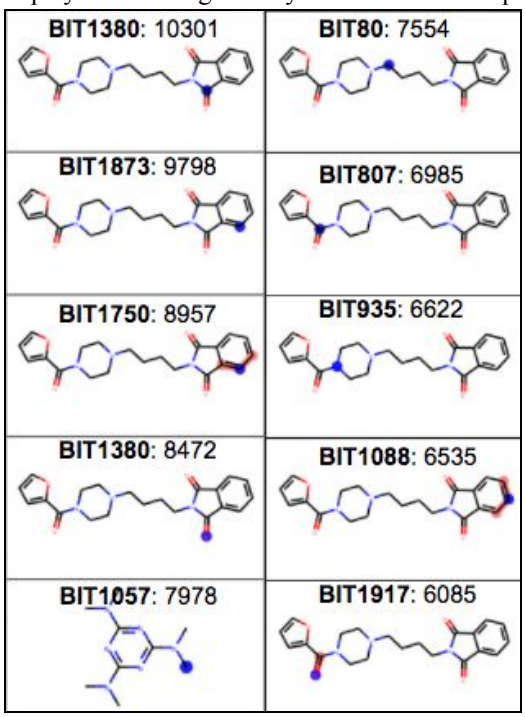


FIGURE 4: Bit Frequency Statistics

Mean	231.520
Standard Deviation	681.523
Minimum	3.000
Maximum	10301.000
25th Percentile	46.000
Median	84.000
75th Percentile	178.000

FIGURE 5: Most Frequent Bits

Fragments are highlighted in red, with the anchor molecule in blue, in the context of fragment molecules from the training corpus. The majority of high-frequency bits have a radius of zero and indicate single atoms with the displayed VSEPR geometry. Individual bit-frequencies are listed following each Id.



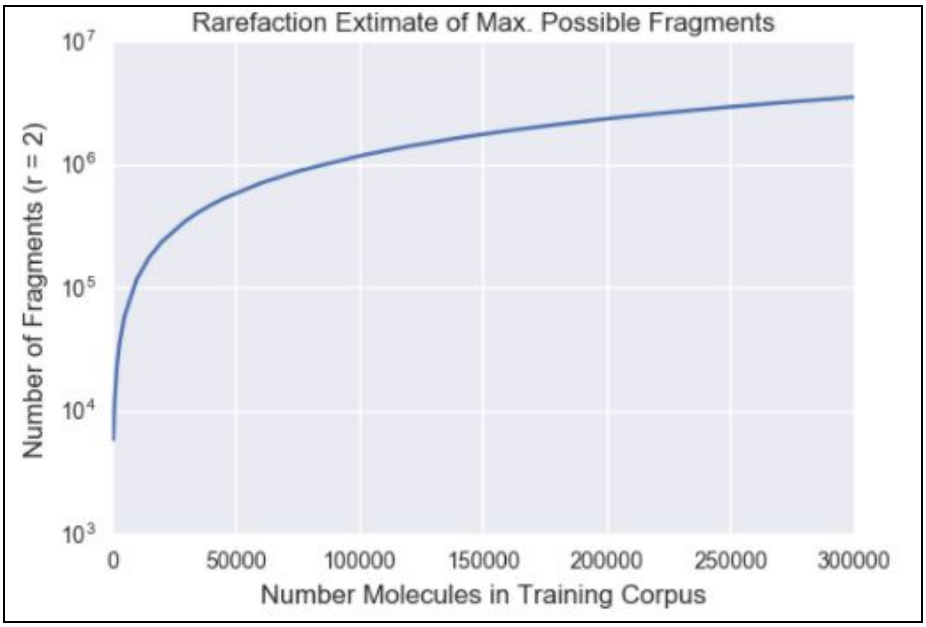


FIGURE 6: Rarefaction plot, estimating the theoretical total of all possible molecular-fragments (of radius = 2) in chemical space as ~3,500,000. This complexity can be captured by *unbiased* corpuses >250,000 molecules.

CODE BLOCK 3: Rarefaction Calculations. PubChem was randomly sampled to generate discrete fragment libraries from corpuses of different sizes. The Number of molecules in each corpus was plotted against the number of respective fragments it generated.

```
# RANDOM PCID GENERATION
import pandas as pd
import numpy as np
from pubchempy import *
from random import randint

db_size = [500, 1000, 2000, 3000, 4000, 5000, 10000, 15000, 20000, 50000,
        60000, 800000, 100000,150000, 200000, 300000]

for y in db_size:
    temp = []
    random_pcids = [randint(10000, 999999) for x in range(1,y+1)]
    for r in random_pcids:
        p = get_properties('CanonicalSMILES', r, namespace=u'cid')
        temp.append(p[0].values()[0])
    filename = str(y) + 'smiles'
    pd.Series(temp).to_csv(filename, index=None)
```

```
# GENERATE A FRAGMENT CATALOG FOR EACH DB SAMPLING AND FIND SIZE OF FRAG SPACE
fName = os.path.join(RDConfig.RDDataDir,'FunctionalGroups.txt')
frag_amts = []

for db in db_sizes[-1:]:
    fparams = FragmentCatalog.FragCatParams(1,6,fName)
    cat = FragmentCatalog.FragCatalog(fparams)
    fcgen = FragmentCatalog.FragCatGenerator()
    temp = db_list[db]

    for s in temp:
        try:
            m = Chem.MolFromSmiles(s)
        except Exception as e:
            pass
        try:
            nAdded=fcgen.AddFragsFromMol(m, cat)
        except Exception as e:
            pass
    size = cat.GetNumEntries()
    t = (db, size)
    frag_amts.append(t)
```

```
# PREPARE PLOT
db_size = [500, 1000, 2000, 3000, 5000, 10000, 15000, 20000, 30000, 35000, 40000, 45000, 60000]
xi = [500, 1000, 2000, 3000, 5000, 10000, 15000, 20000, 30000, 35000, 40000, 45000, 60000,
      75000, 100000, 120000, 140000, 160000, 180000, 200000, 220000, 230000, 250000, 270000, 300000]

z = np.polyfit(db_size, np.log(yt), 1)

print z

[  4.39566404e-05   1.17723326e+01]

zx = lambda x: 4.39566404e-05 + 11.772332583482433*x

new = [zx(i) for i in xi]

ax = plot(xi, new)
```

Due diligence was taken to ensure that the complexity of the training set captured the diversity of the unmitigated chemical "universe." This analysis is key to constructing a training corpus that will minimize bias. ***Figure 6*** exhibits a plot to estimate the theoretical amount of "all" molecular fragments on Earth. This is accomplished through means of rarefaction, a common technique for extrapolating complexity[7]. These calculations (***Code Block 3***) determined that the training set's fragment-space (~300,000 fragments) does not converge to the upper bound, and fails to serve as a basis for representing the entire chemical-fingerprint space (~3,500,000 fragments) conjectured above. By this metric, the complexity of the training corpus does not fully encapsulate the exhaustive observation of any possible molecular fingerprint, and combinations thereof, within the scope of activity classification. To improve robustness, an additional model was scaled to a training set of 250,000 compounds. This model has, however, been withheld for intellectual property concerns.

FEATURE ENGINEERING

To better understand bit importance and perform feature engineering, bitwise Information Gain (Information_Gain = Prior_Entropy - Subsequent_Entropy) was used to identify the most informative and predictive fragments of the active class (***Code Block 4***). In this scenario, Shannon Entropy (H = - p(a)*log(p(a)) - p(b)*log(p(b))) was used to observe the amount of information gained by the incremental inclusion of each fragment as a feature for recapitulating the screen dataset's compound activity-distribution. Information is therefore gained when the probability distribution predicted by a previous set of features improves and includes fewer misclassified points due to the inclusion of a new feature. In other words, the distribution generated by the updated feature-set decreases in Shannon Entropy, such that the projections are more "pure" and accurate in depicting the true activity-distribution of the training set. With this in mind, information gain was used to ascertain the top-ten fragments that best correspond with compound activity(***Figure 7***).

These results, however, did not incur feature-space compression, as negligible activity-correlations (i.e. coefficients of approximately zero) were observed across all features within the information-gain truncated set. Further attempts at feature engineering included PCA, but were equally unsuccessful in reducing bit-vector dimensionality, seeing as no set of principal components was able to describe enough variance or structure within the training corpus. For this reason, all 2048 bits/fragments were retained to serve as features in subsequent models. These null results are not recreated in this report, but are documented under the repository's "Code" folder, in the "TrainingSpace" notebook.

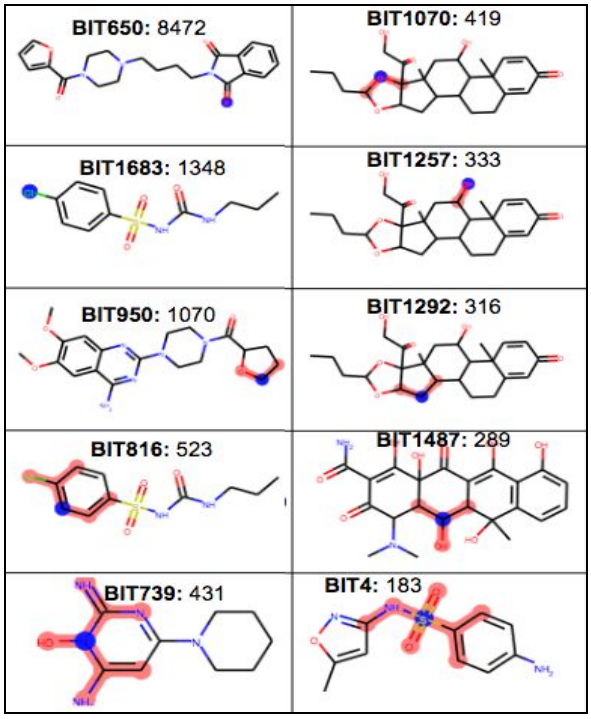
CODE BLOCK 4: RdKit Retrieval for Top-Bits by Information-Gain

```
# ID TOP PREDICTIVE FRAGMENTS BY INFO GAIN
from rdkit.ML.InfoTheory import InfoBitRanker
from rdkit.Chem import FragmentCatalog
from rdkit import Chem
from rdkit import RDConfig

ranker = InfoBitRanker(2048,2)
acts = [x for x in chem_lib.Activity]

for i,fp in enumerate(train_bit):
    act = int(acts[i])
    ranker.AccumulateVotes(fp,act)

# Collect Top 20 Bits
top20 = ranker.GetTopN(20)
top20_bitids = []
for id,gain,n0,n1 in top20:
    top20_bitids.append(id)
    print(int(id),'%.3f'%gain,int(n0),int(n1)) # bitId, infoGain, nInactive, nActive
```

```
# FIND EXAMPLE MOLECULES FOR THE TOP 20 BITS BY INFO GAIN
top_bitids = [int(x) for x in top_bitids]

topbit_ix = []
topbit_ex = []
for i in top_bitids:
    mol_ex_ix = train_bit_df.Bit_ID[train_bit_df.Bit_ID==i].index
    topbit_ix.append(mol_ex_ix)
    hit_pccid = train_bit_df.PC_CID.ix[mol_ex_ix]
    topbit_ex.append(hit_pccid)
```

FIGURE 7: Top 10 Bits by Information Gain

Fragments are highlighted in red, with the anchor molecules in blue, in the context of exemplary molecules from the training corpus.



MODEL DEVELOPMENT

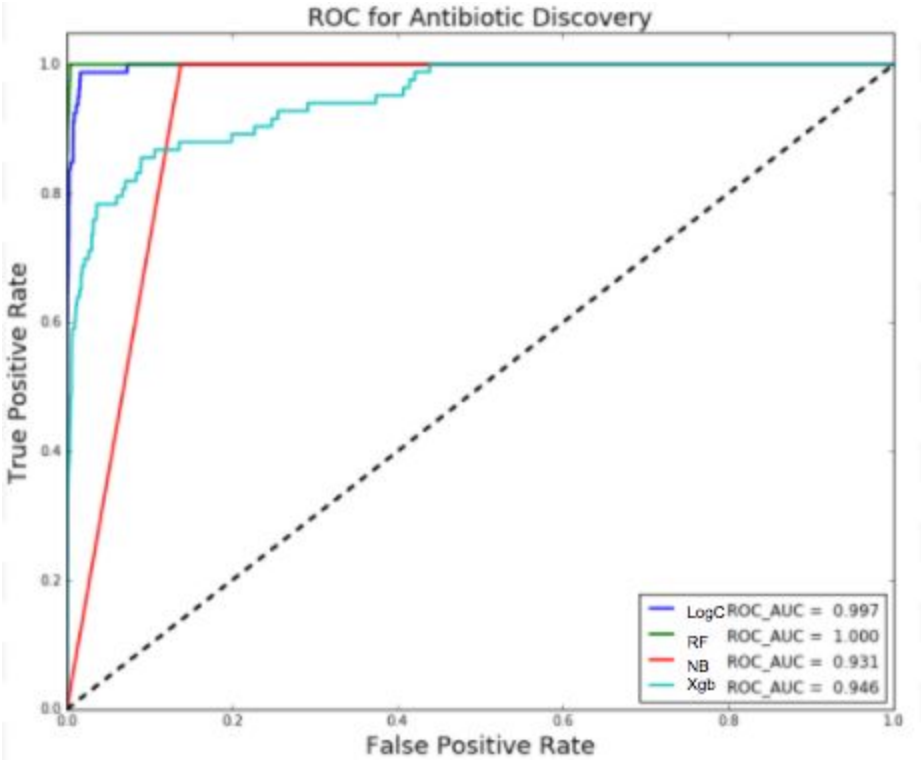
Antitubercular drug-discovery is, in essence, a binary classification problem--either a compound is “Active,” killing over 90% of bacteria in the reported screen, or “Inactive.” Due to the subsequent nature of this problem, a variety of binary classification algorithms were explored to derive a superlative predictor. The findings are as follows.

To develop an elementary insight into the cheminformatic instructiveness of the training set, a logistic classifier was employed first and with favorable success. Additional models were fashioned and performance metrics are summarized in ***Figure 8***, with the generative code for each model supplied in the project repository. Overall, a Random Forest Classifier emerged superior and it’s code and hyperparameter optimization are described in the subsequent Model Optimization section.

FIGURE 8: Model Statistics. Summarized model statistics, as derived from a stratified, train-test-split with 33.333% of the training corpus reserved for testing. See “ModelValidation” notebook for complete details.

Model	Recall	Precision	F1	Accuracy	ROC_AUC
Logistic Classifier	0.5732	0.7917	0.7287	0.9900	0.9970
Random Forrest	0.8516	0.9205	0.9139	0.9963	1.0000
Naive Bayes	1.0000	0.5691	0.2428	0.8569	0.9310
Gradient-Boosted LGC	0.2333	0.6256	0.3784	0.9802	0.9462

FIGURE 9: Model ROC-AUC Curves



MODEL OPTIMIZATION

Associated code for the final model is exhibited below (***Code Block 5***), with the optimized parameters in ***Figure 10***. In overview, SciKit-klearn[8] was utilized to generate a Random Forest Classifier, whose hyperparameters were further optimized through GridSearch over an exhaustive combination of values. Model quality was assessed through the same statistics listed in ***Figure 8*** and scrutinized over a 5-fold, Stratified Cross-Validation. Favorable precision and recall scores were prioritized in comparing each model, due to class imbalance within the training corpus and desire to minimize false-positive rates.

CODE BLOCK 5: Derivation of finalized model, as performed with SciKit-Learn over a twice cross-validated Grid Search for hyperparameter optimization.

```
# DEFINE PARAMS FOR GRIDSEARCH
params = {
    'n_estimators': [100, 250, 500, 750],
    'criterion': ['entropy'],
    'max_features': ['auto'],
    'random_state': [1024]

}

# DEFINE CV SCHEME
sss = StratifiedShuffleSplit(y, n_iter=2)
# Stratified-Shuffle-Split to ensure train data preserves class imbalance in cv

# DEFINE YOUR ESTIMATOR
rf = RandomForestClassifier()

# DEFINE GRIDSEARCH
gsearch = GridSearchCV(rf, param_grid=params, verbose=2, cv=sss, scoring='roc_auc')

# EXECUTE GRIDSEARCH
gsearch.fit(x, y)
```

```
# EXTRACT BEST FEATURES TO DEFINE/FIT OPTIMAL XGBLOGREG

print "BEST PARAMS:" , gsearch.best_params_
rf_best.set_params(**gsearch.best_params_)
```

FIGURE 10: Optimized Random Forest Model. The finalized model is too large to display as a complete graphic (Note: there are 100 trees in the forest, as n_estimators = 100, with 100 bits considered/bagged per node, max_features = 100). The decision diagram can rerendered and navigated in full within the "ModelValidation" notebook.

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
            max_depth=None, max_features=100, max_leaf_nodes=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
            oob_score=True, random_state=1024, verbose=0, warm_start=False)
```

RESULTS

CHEMICAL SCREENING OUTCOMES

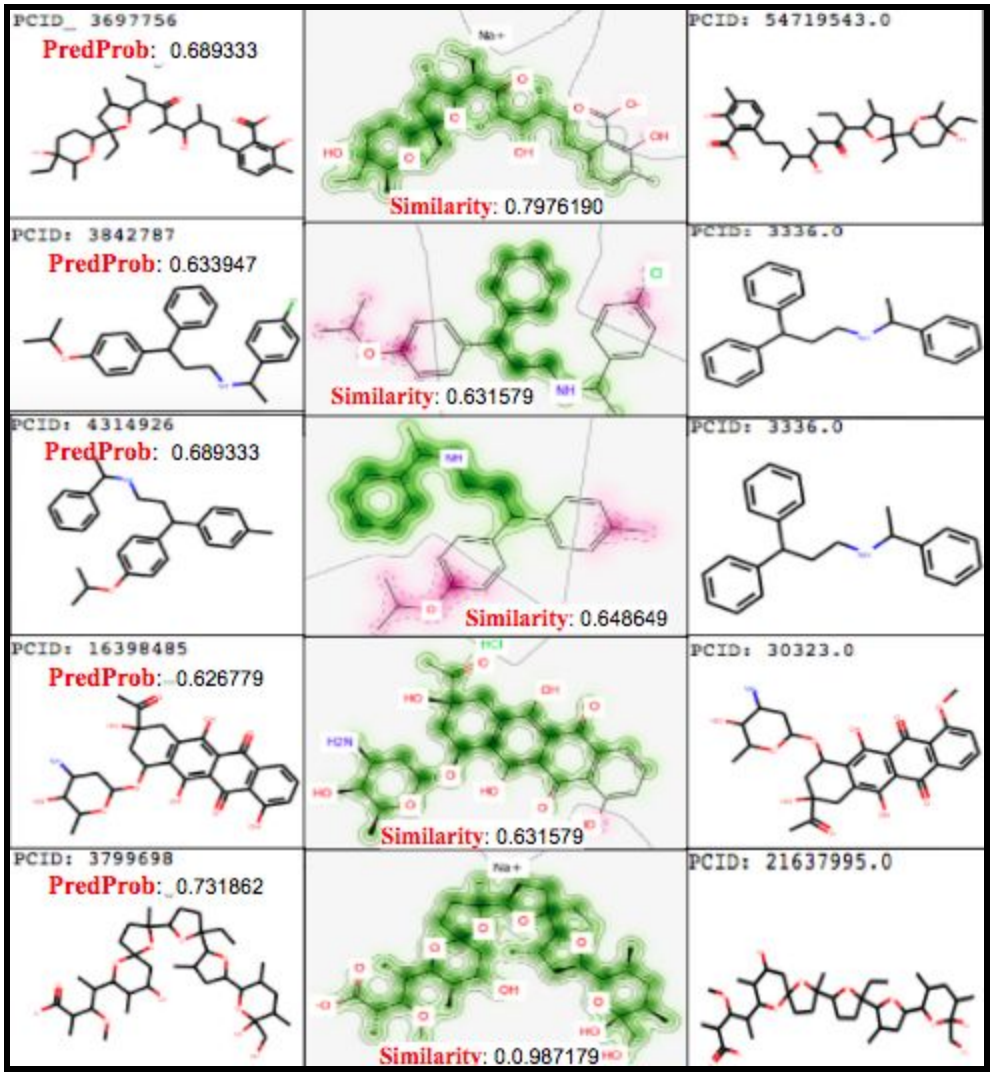


FIGURE 11: Predicted Antitubercular Candidates. Of the 29 predicted, 5 example compounds are listed above. Kekule structures for each are depicted in the leftmost column, then superimposed with respective top matches from training-set (rightmost column) to generate electron similarity maps (center column).

The final, Random Forrest Classifier was used to screen the uncharacterized natural-product library and returned 29 drug candidates. Each hit was mapped to its closest match from the positive training subset (see ***Code Block 6***), then superimposed to generate a similarity map (***Figure 11***). These figures implicate specific features that weighted toward the probabilistic activity-prediction determined by the model. Fragments that contribute to similarity with the active class are highlighted in green and the fragments that detract from similarity are accentuated in magenta. Molecular fragments that had no effect on similarity are ubiquitous atomic motifs, frequent across all classes in the corpus, and are left unaltered, encased within grey topological

markings. Further details for deciphering compound similarity maps are elaborated in Riniker et al[9].

Though high similarities between candidate and active training molecules bodes well for their activity, raw Tanimoto Similarity scores (aka. Jaccard Distance between molecular bit/fragment vectors) are notoriously hard to interpret, especially as they are incomparable across separate corpuses. However, the electron-density similarity maps above, allow for unequivocal demonstration of homologous structures between molecules. One can thereby leverage the fact that form-fits-function, such that the cumulative electrostatic surface of the molecules, from which chemical activity and binding proclivities are defined, must also be similar.

CODE BLOCK 6: Candidate Compound Visualisation Code.
See "ScreenValidation" notebook for further details.

```
# GENERATE FINGER PRINTS FOR ALL POSITIVE CLASS COMPOUNDS
hit_mol, hit_bit, hit_fp = make_fingerprints(np_hit.Smiles)
train_mol, train_bit, train_fp = make_fingerprints(chem_lib.Smiles)
```

```
# DEFINE FX: GENERATE HIT KEKULE STRUCTURES AS SVG

def moltosvg(smi,molSize=(200,125), kekulize=True):
    from rdkit.Chem import rdDepictor
    from rdkit.Chem.Draw import rdMolDraw2D

    mol = Chem.MolFromSmiles(smi)
    mc = Chem.Mol(mol.ToBinary())
    if kekulize:
        try:
            Chem.Kekulize(mc)
        except:
            mc = Chem.Mol(mol.ToBinary())
    if not mc.GetNumConformers():
        rdDepictor.Compute2DCoords(mc)
    drawer = rdMolDraw2D.MolDraw2DSVG(molSize[0],molSize[1])
    drawer.DrawMolecule(mc)
    drawer.FinishDrawing()
    svg = drawer.GetDrawingText()
    return svg.replace('svg:','')

def show_struct(smi):
    from PIL import Image
    from rdkit.Chem.Draw import IPythonConsole
    from IPython.display import SVG
    from rdkit.Chem import PandasTools
    PandasTools.RenderImagesInAllDataFrames(images=True)

    svg = moltosvg(smi)
    return SVG(svg)
```

```
# CREATE KEKULE STRUCTURES FOR ALL HTS
from rdkit.Chem.Draw import MolsToGridImage
hitimg = MolsToGridImage(hit_mol,molsPerRow=4,subImgSize=(200,200),legends=["PC_ID:" + str(x) for x in np_hit.Id])
hitimg
```

```
# PLOT SUPERPOSITION GRAPH FOR EACH HIT AGAINST ITS MOST SIMILAR MOLECULE FROM TRAINING SET
from rdkit import Chem
from rdkit.Chem.Draw import MolsToGridImage
from rdkit.Chem import DataStructs
from rdkit.Chem.Draw.SimilarityMaps import GetSimilarityMapForFingerprint
from rdkit.Chem.Draw.SimilarityMaps import GetMorganFingerprint

# Calculate bulk Tanimoto Similarity scores
temp = []
for x in range(len(hit_bit)):
    sims = DataStructs.BulkTanimotoSimilarity(hit_bit[x], train_bit)
    temp.append(sims)

for f in range(len(hit_mol)):
    max_sim = max(temp[f])
    mi = temp[f].index(max_sim)

    fig, maxweight = GetSimilarityMapForFingerprint(train_mol[mi], hit_mol[f],
                        lambda m,idx: GetMorganFingerprint(m, radius=2, atomId=idx, fpType='count'),
                        metric=DataStructs.TanimotoSimilarity)
    print "Max Match for hit No. %d: %d, %f" % (f, mi, max_sim)
```

However provocative, these compounds require further validation to discern pharmaceutical viability. This shortcoming is mainly due to experimental design, for the model is fundamentally biased, having been trained solely on *in vitro* data. While the majority of predicted candidates likely confer antitubercular activity, there is no ensuring their pharmacokinetic applications *in vivo*, that is, within the context of the human metabolic backdrop. For instance, within a host, a drug may very well become ineffective--perhaps becoming sequestered by off-target human proteins with superior binding affinities, or metabolised into a new, inactive form by ambient conditions. Notwithstanding, mouse-modeled *in vivo* screening data was recently acquired from The CDD and will be ensembled with the *in vitro* predictor into a pipeline to account for *in vivo* efficacy. To further improve performance, the model will next be scaled to a training set of 250,000 compounds, with further intensive hyperparameter optimization. This improved model will incur less bias, seeing as its training corpus better captures the true complexity of the chemical universe (~3,500,000 fragments at radius = 2), as conjectured by rarefaction extrapolation (***Figure 6***), and observes a more vast selection of molecular fragments in the context of antitubercular activity. Though version 2.0 of this model has been implemented, its findings and details are withheld for intellectual property concerns.

REFERENCES

1. WHO. (2016, October). WHO | Tuberculosis. Retrieved from http://www.who.int/mediacentre/factsheets/fs104/en/
2. Center for Disease Control. (2016, September 8). Biggest Threats| Antibiotic/Antimicrobial Resistance | CDC. Retrieved from http://www.cdc.gov/drugresistance/biggest_threats.html
3. Collaborative Drug Discovery, Inc. (n.d.). CDD Vault: Modern Drug Data Management. Retrieved from https://www.collaborativedrug.com/cdd-vault
4. NIH: Southern Research Institute. (2008, June 26). AID 1332 - High Throughput Screen to Identify Inhibitors of Mycobacterium tuberculosis H37Rv - PubChem. Retrieved from https://pubchem.ncbi.nlm.nih.gov/bioassay/1332
5. Landrum, G. (2016). Module rdMolDescriptors. Retrieved October 11, from http://www.rdkit.org/docs/api/rdkit.Chem.rdMolDescriptors-module.html#GetMorganFingerprintAsBitVect
6. InterBioScreen Ltd. (n.d.). DOWNLOAD DATABASES. Retrieved October 30, 2016, from http://www.ibscreen.com/search.shtml
7. CAYUELA, L., GOTELLI, N. J., & COLWELL, R. K. (n.d.). Ecological and biogeographic null hypotheses for comparing rarefaction curves. Retrieved from https://www.researchgate.net/publication/282744514_Ecological_and_biogeographic_null_hypotheses_for_comparing_rarefaction_curves

8. Buitinik, L. (2013, September 1). API design for machine learning software: experiences from the scikit-learn project. Retrieved November 7, 2016, from https://arxiv.org/abs/1309.0238

9. Riniker, S., & Landrum, G. (2013, May). Similarity maps - a visualization strategy for molecular fingerprints and machine-learning methods. Retrieved from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3852750/pdf/1758-2946-5-43.pdf