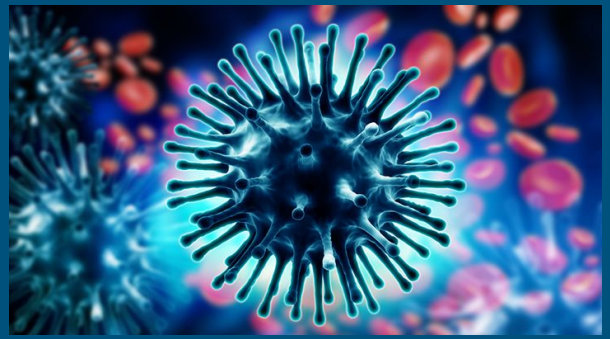
Machine Learning vs. The Flu

Rohan Koodli
PyData SF, August 2016
@RohanKoodli; github.com/RK900

About Me

- Entering 11th grade this fall
- Been programming for the past three years now, mostly in Python (also Java, R)
- I like scientific programming, machine learning, and data analytics

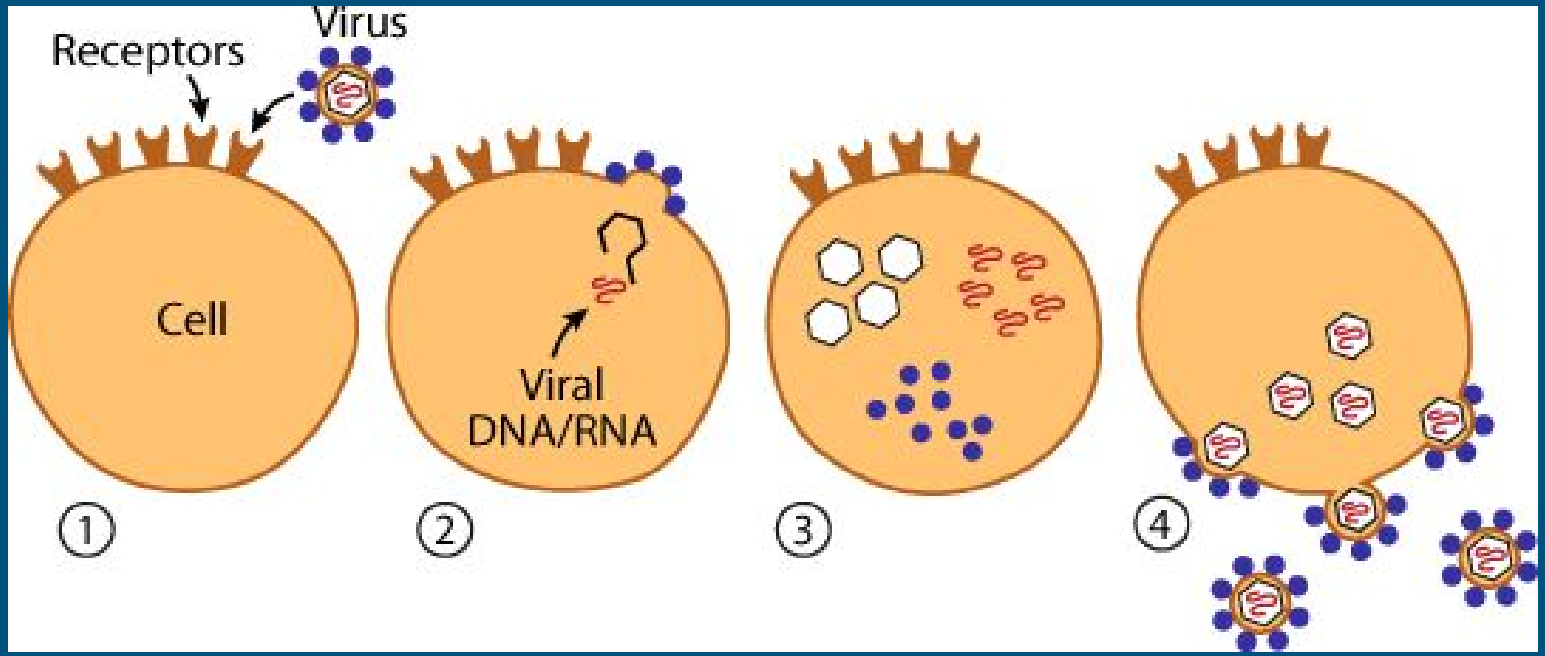
The Problem - Influenza



- Millions affected annually
- Take a vaccine every year, due to the flu’s rapid mutations
- Current methods of creating flu vaccines aren’t very efficient
- Scientists simply vote on what antibodies go in the vaccine (has been called “questionable”)
- Sometimes the vaccines don’t work for every strain (e.g., 2009 H1N1)

Goal: Create a more effective way to create vaccines

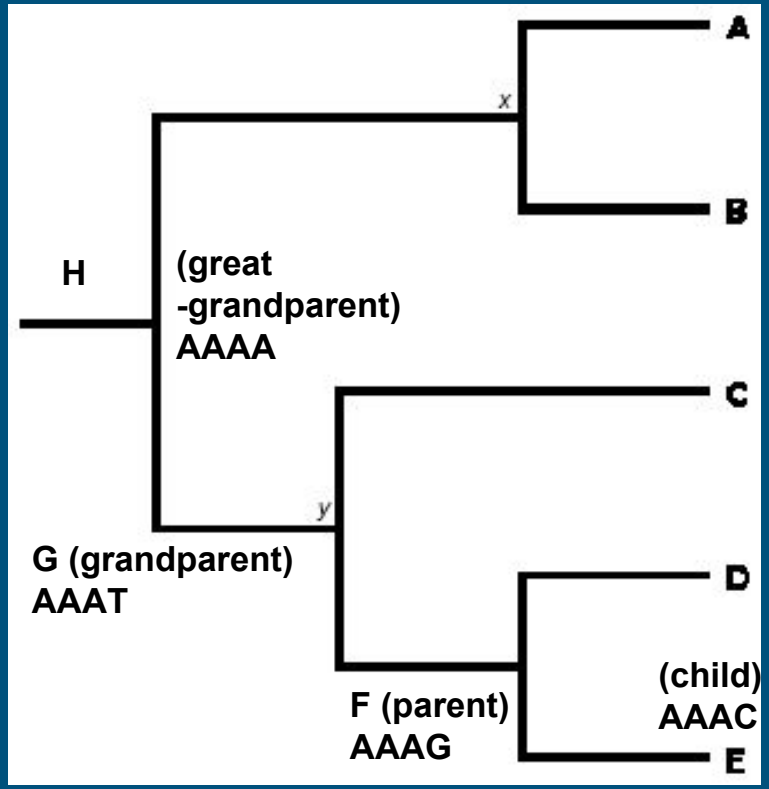
How the flu infects a host



My Approach

- Predict future genetic sequence of the flu, so that scientists can analyze it and create vaccines tailored towards that specific sequence
- Machine learning
- Phylogenetics (study of relationships between individuals on an evolutionary tree)

A Phylogenetic Tree



My Setup

- Obtain data from the Influenza Research Database (IRDB) flu database
- Read in the flu genetic sequence of HA and NA
- Train a machine learning model (final model determined later in the project) on the phylogenetic (grandparent-parent-child) relationships between flu strains and evaluate accuracy

However...

- I wasn't sure how accurate a homemade ML algorithm would be
- Which ML algorithm to use?
- How to measure accuracy?
- How to read in a FASTA file (Common file format for biological data)?

Biopython to the rescue!

The flu prediction algorithm

- I did some research and found that a decision tree-based algorithm would be the best fit
  - Phylogenetic tree maps to a decision tree

Needless to say, there were many problems

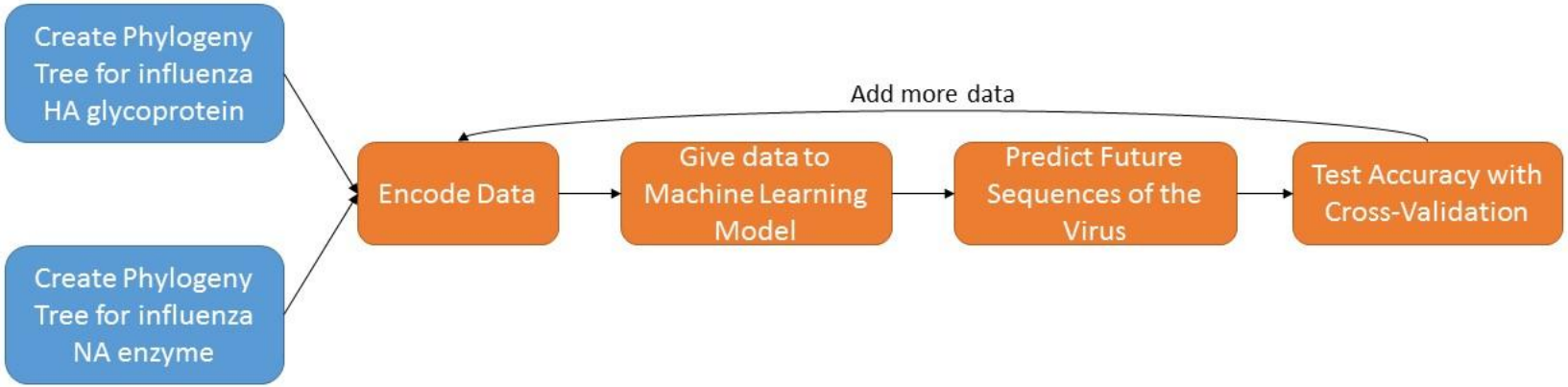
- Only was a classification algorithm (regression would be more ideal)
- Underfitting

scikit-learn to the rescue!

The Setup (updated)

- Obtain data from IRDB
- Read in the flu genetic sequence FASTA file using Biopython
- Encode data into numbers instead of ATGC
- Train a machine learning model on the parent-child relationships between flu strains using scikit-learn and evaluate accuracy
- Test accuracy with cross-validation and R^2 score

Overview of Algorithm



Biopython

- Simple one-liner to read FASTA file

```
new = list(SeqIO.parse('/Data-Files/HA-100.fasta','fasta'))
```

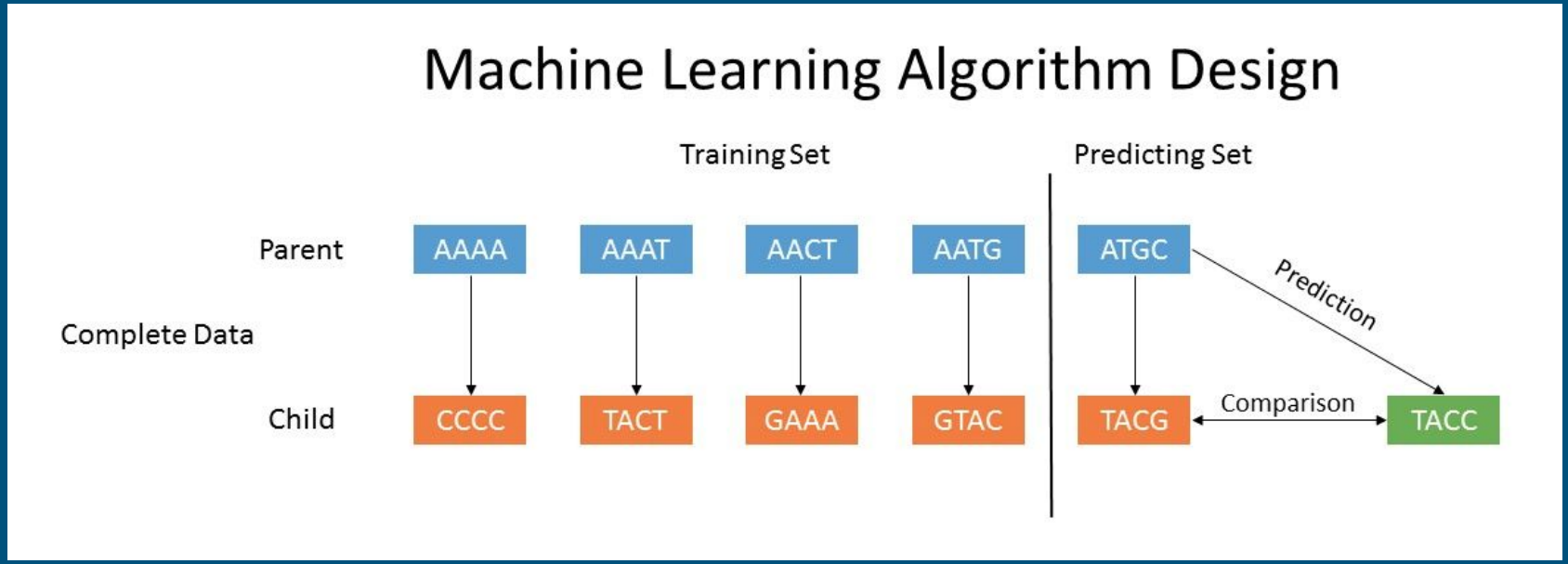
The encoding method

```
def encoding(sequence):
    encoded = []
    for i in sequence: #turns base pairs into numbers
        if i == 'A':
            encoded.append('1')
        if i == 'T':
            encoded.append('2')
        if i == 'G':
            encoded.append('3')
        if i == 'C':
            encoded.append('4')

    #print encoded prints fine

    seq = []

    for i in range(len(encoded)):
        if i%15 == 0:
            a = encoded[i:i+15]
            a = ''.join(a)
            a = float(a)
            seq.append(a)

    return seq
```

```
class SimpleDecisionTree:

    _tree = {}  # this instance variable becomes accessible to class methods via self._tree

    def __init__(self):
        # this is where we would initialize any parameters to the SimpleDecisionTree
        pass

    def fit(self,
            instances,
            candidate_attribute_indexes=None,
            target_attribute_index=0,
            default_class=None):
        if not candidate_attribute_indexes:
            candidate_attribute_indexes = [i
                                           for i in range(len(instances[0]))
                                           if i != target_attribute_index]
        self._tree = self._create_tree(instances,
                                       candidate_attribute_indexes,
                                       target_attribute_index,
                                       default_class)

    def _create_tree(self,
                     instances,
                     candidate_attribute_indexes,
                     target_attribute_index=0,
                     default_class=None):
        class_labels_and_counts = Counter([instance[target_attribute_index]
                                           for instance in instances])
        if not instances or not candidate_attribute_indexes:
            return default_class
        elif len(class_labels_and_counts) == 1:
            class_label = class_labels_and_counts.most_common(1)[0][0]
            return class_label
        else:
            default_class = simple_ml.majority_value(instances, target_attribute_index)
            best_index = simple_ml.choose_best_attribute_index(instances,
                                                               candidate_attribute_indexes,
                                                               target_attribute_index)
            tree = {best_index:{}}
            partitions = simple_ml.split_instances(instances, best_index)
            remaining_candidate_attribute_indexes = [i
                                                     for i in candidate_attribute_indexes
                                                     if i != best_index]
            for attribute_value in partitions:
                subtree = self._create_tree(
                    partitions[attribute_value],
                    remaining_candidate_attribute_indexes,
                    target_attribute_index,
                    default_class)
                tree[best_index][attribute_value] = subtree
            return tree

    def predict(self, instances, default_class=None):
        if not isinstance(instances, list):
            return self._predict(self._tree, instance, default_class)
        else:
            return [self._predict(self._tree, instance, default_class)
                    for instance in instances]

    def _predict(self, tree, instance, default_class=None):
        if not tree:
            return default_class
        if not isinstance(tree, dict):
            return tree
        attribute_index = list(tree.keys())[0]  # using list(dict.keys()) for Py3 compatibility
        attribute_values = list(tree.values())[0]
        instance_attribute_value = instance[attribute_index]
        if instance_attribute_value not in attribute_values:
            return default_class
        return self._predict(attribute_values[instance_attribute_value],
                             instance,
                             default_class)
```

```
from sklearn import ensemble
rfr = ensemble.RandomForestRegressor(n_estimators=20)
rfr.fit(X,y)

rfrscores = cross_validation.cross_val_score(rfr,X,y,cv=2)
print 'Random Forests',rfrscores
print("Average Accuracy: %0.2f (+/- %0.2f)" % (rfrscores.mean()*100, rfrscores.std() *100))
```

K-fold cross-validation



cross-validation

```
150 y_pred_rfr = rfr.predict(X_test)
151 print 'Random Forests R2 score:', metrics.r2_score(y_test,y_pred_rfr,multioutput='uniform_average')
```

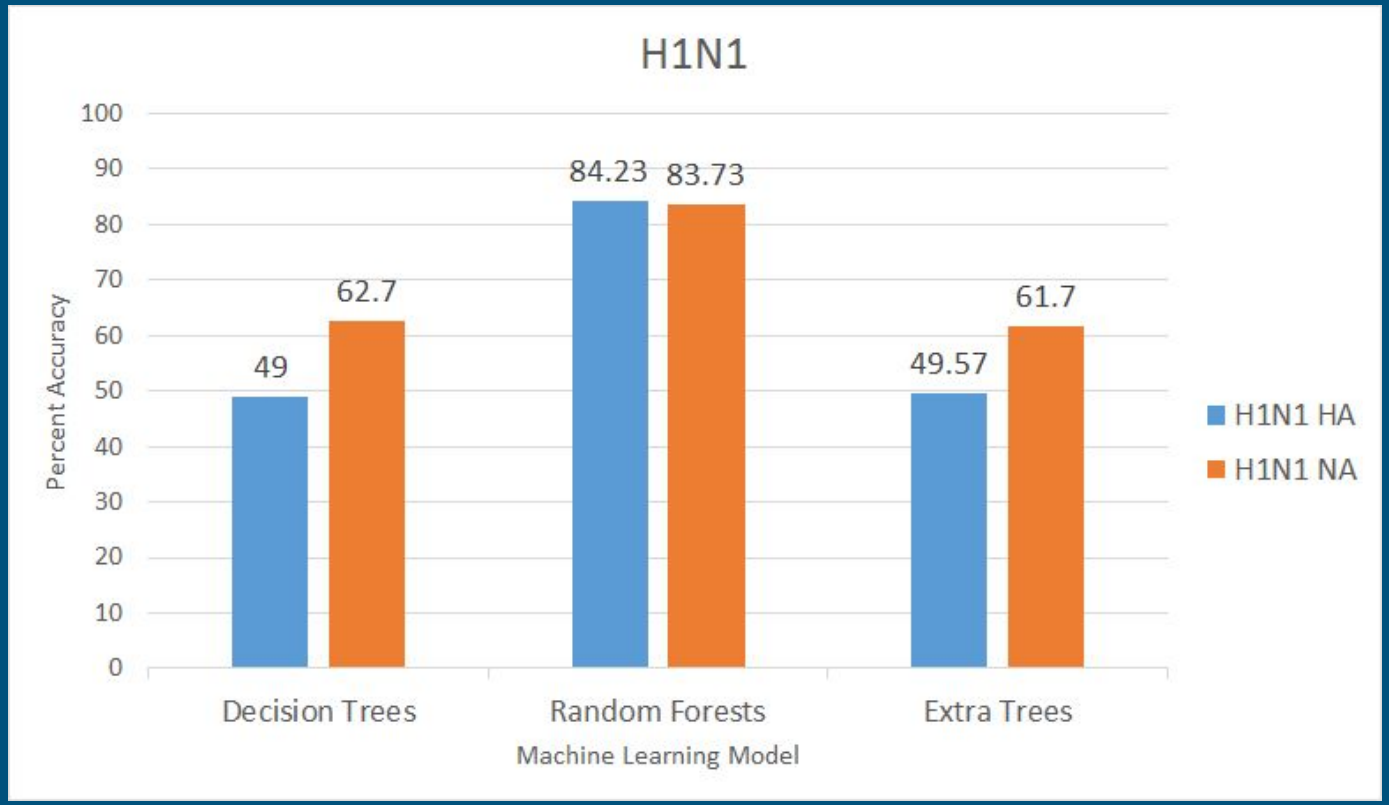
R^2 score

Drawbacks (aka a Wish List for features)

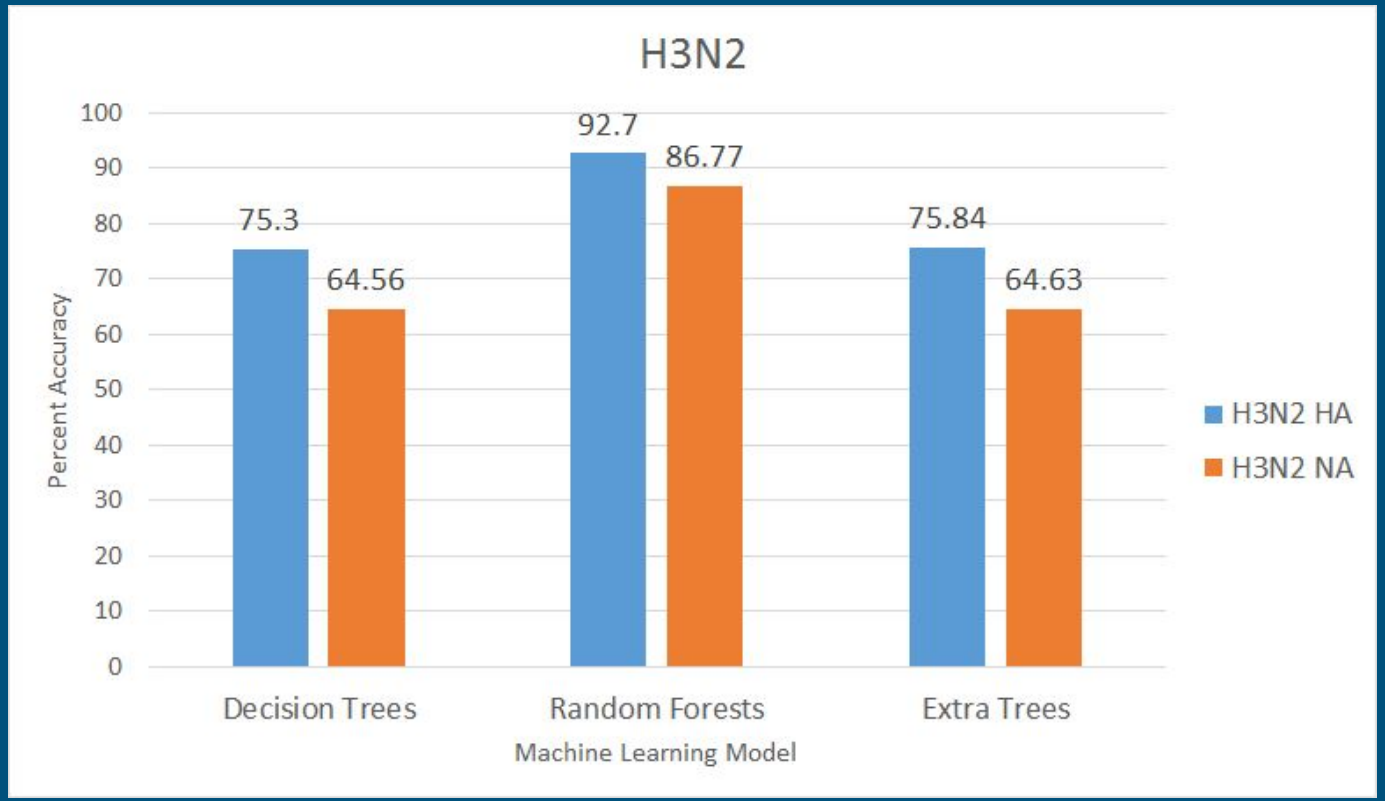
- Couldn’t train on the grandparent-parent-child relationships using scikit-learn
- Couldn’t process letters or chars
- Couldn’t hold many significant digits

Results

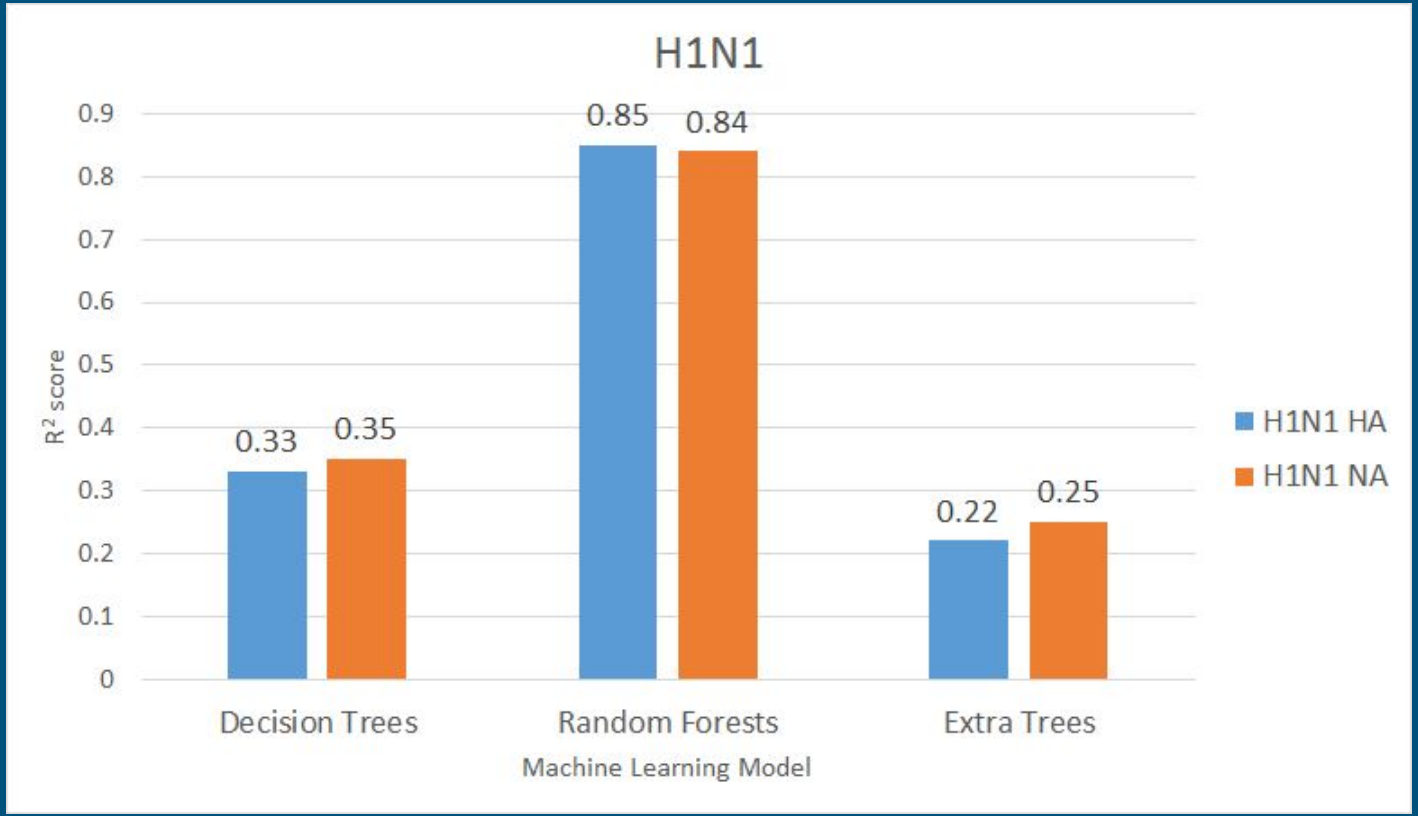
H1N1 - cross-validation scores



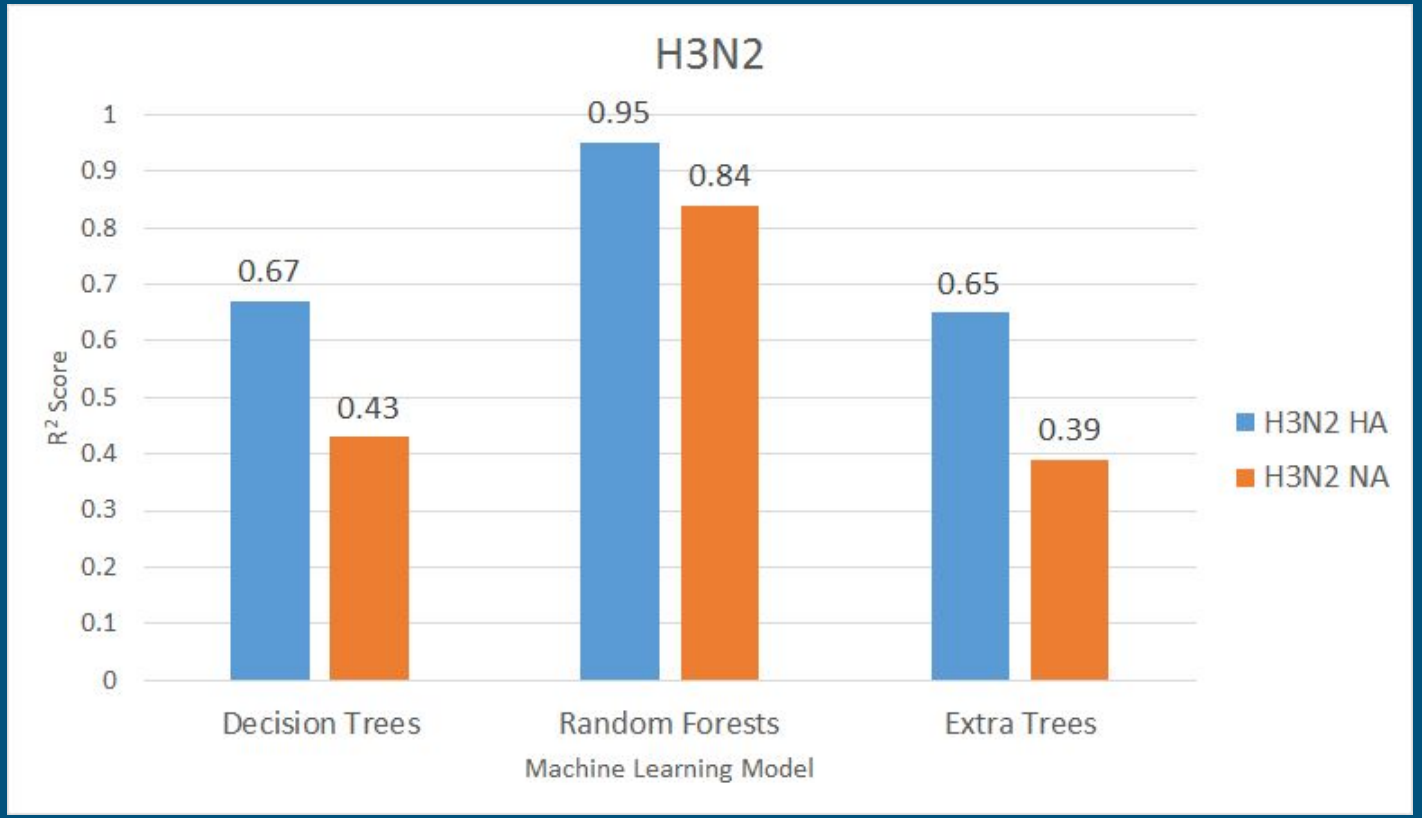
H3N2 - cross-validation scores



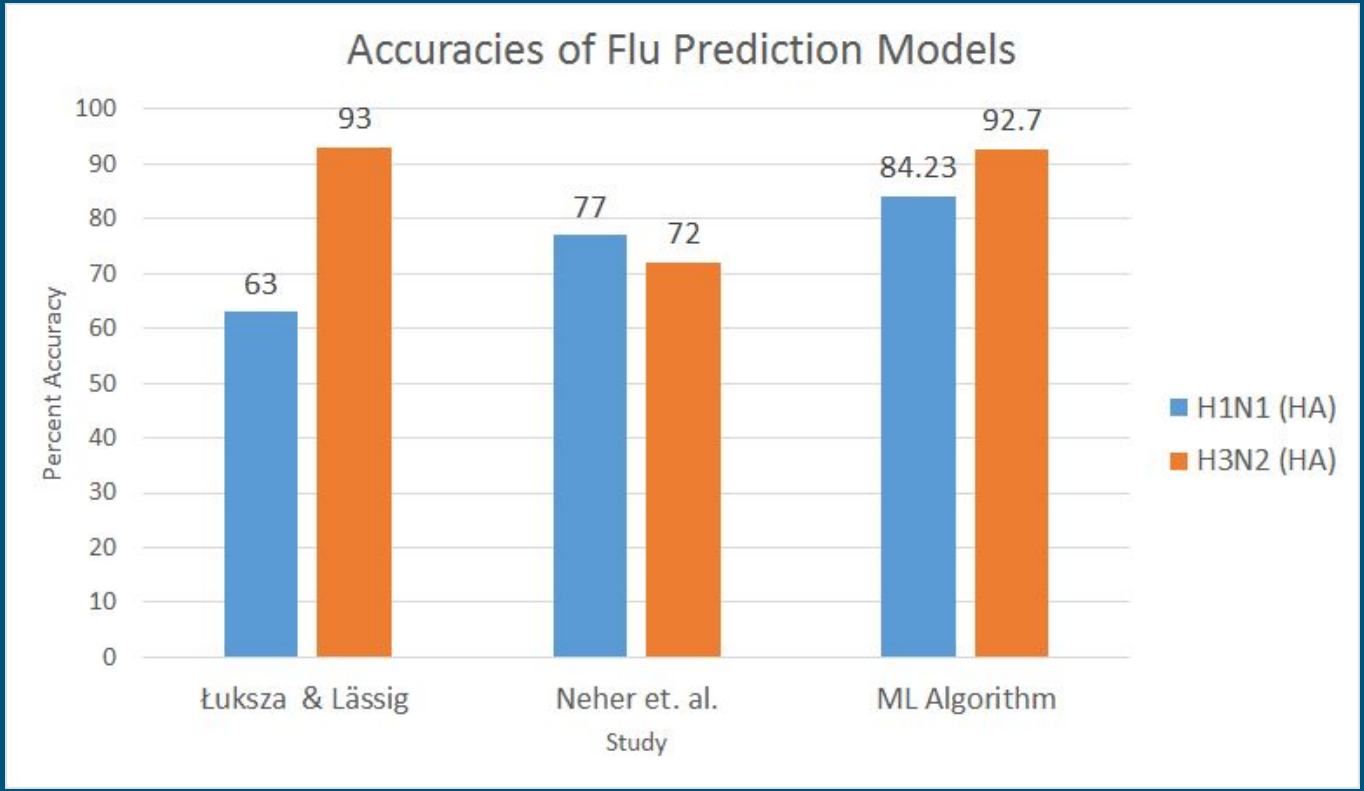
H1N1 - R^2 scores



H3N2 - R^2 scores

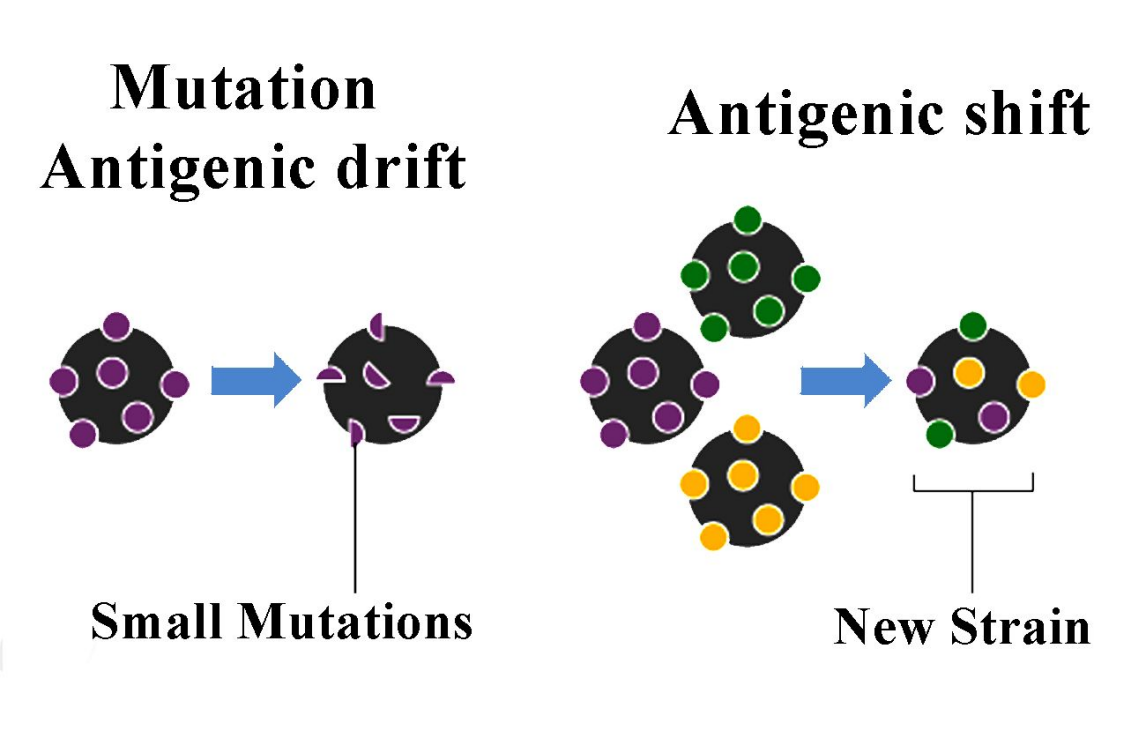


Comparison with previous studies



Data Interpretation

Flu Mutations



Why Random Forests performed well

- Random Forests creates ensemble of trees, lowering variance, less importance to antigenic shift
- Decision Trees gave too much importance to shift changes
- Extra Trees gave too little importance to shift changes

Summary

- Better mechanism for influenza vaccines
- Used Biopython and scikit-learn to simplify + more accuracy
- Encouraging results

Thank you!

Twitter: @RohanKoodli
GitHub: github.com/RK900
This project: github.com/RK900/Flu-Prediction