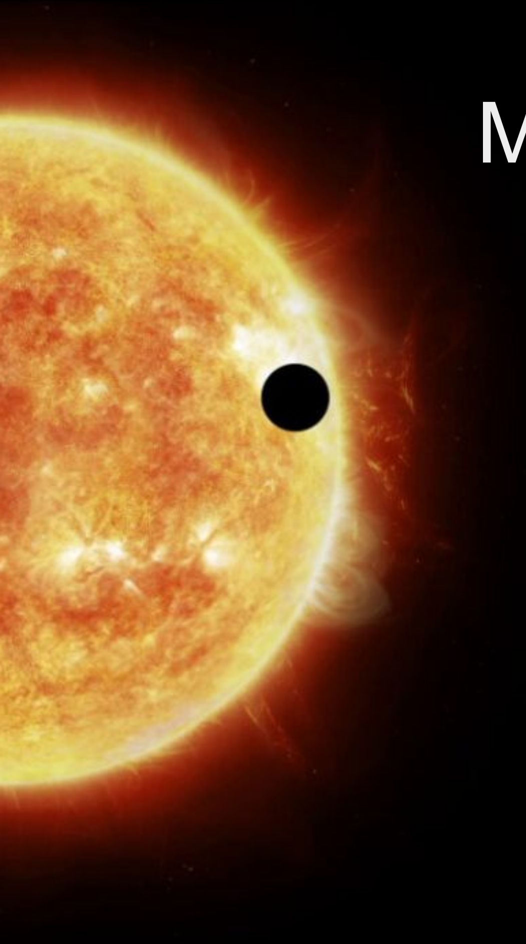


Machine Learning Approaches to the Detection of Exoplanet Transits

Anna Zuckerman, Leah Zuckerman, Ashutosh Gandhi, and Andrew Floyd

CSCI 4502/5502 Summer 2023

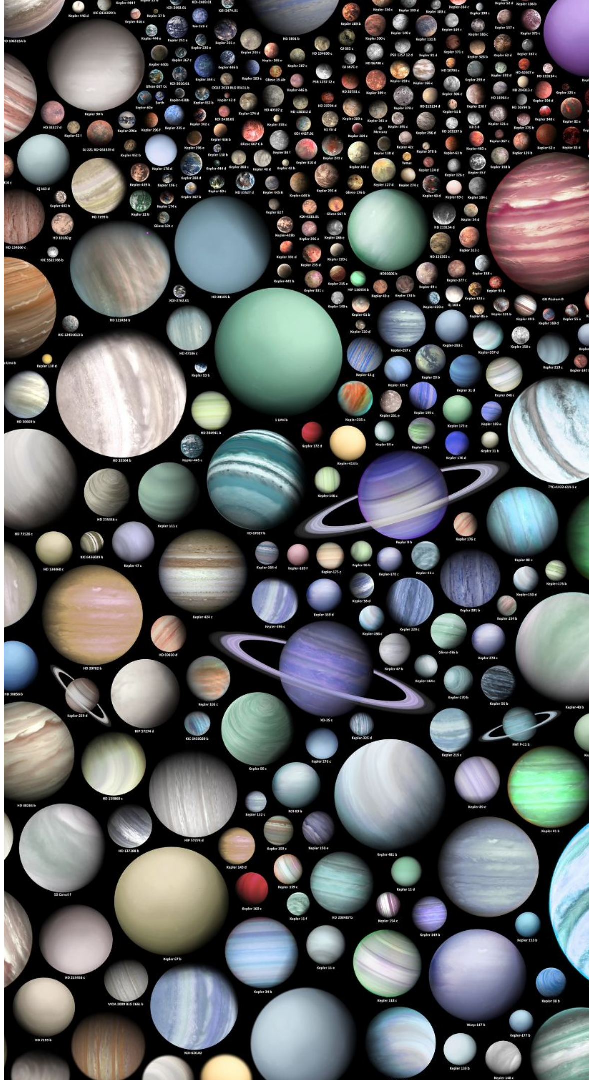
How can we find exoplanets?

Exoplanet = planet orbiting a star other than the sun

Only in science fiction until 1992!

- Since then, thousands discovered

Artists representation!



Graphic: Martin Vargic

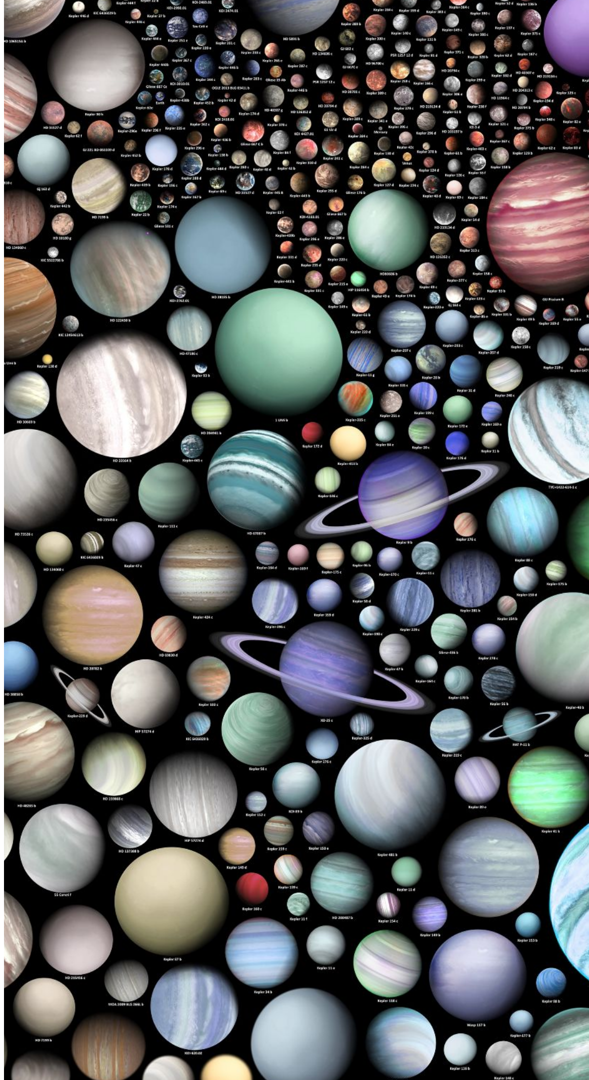
How can we find exoplanets?

Exoplanet = planet orbiting a star other than the sun

Only in science fiction until 1992!

- Since then, thousands discovered

Discovery Method	Number of Planets
Astrometry	2
Imaging	67
Radial Velocity	1048
Transit	4092
Transit timing variations	25
Eclipse timing variations	17
Microlensing	200
Pulsar timing variations	7
Pulsation timing variations	2
Orbital brightness modulations	9
Disk Kinematics	1



Graphic: Martin Vargic,Table: NExSci

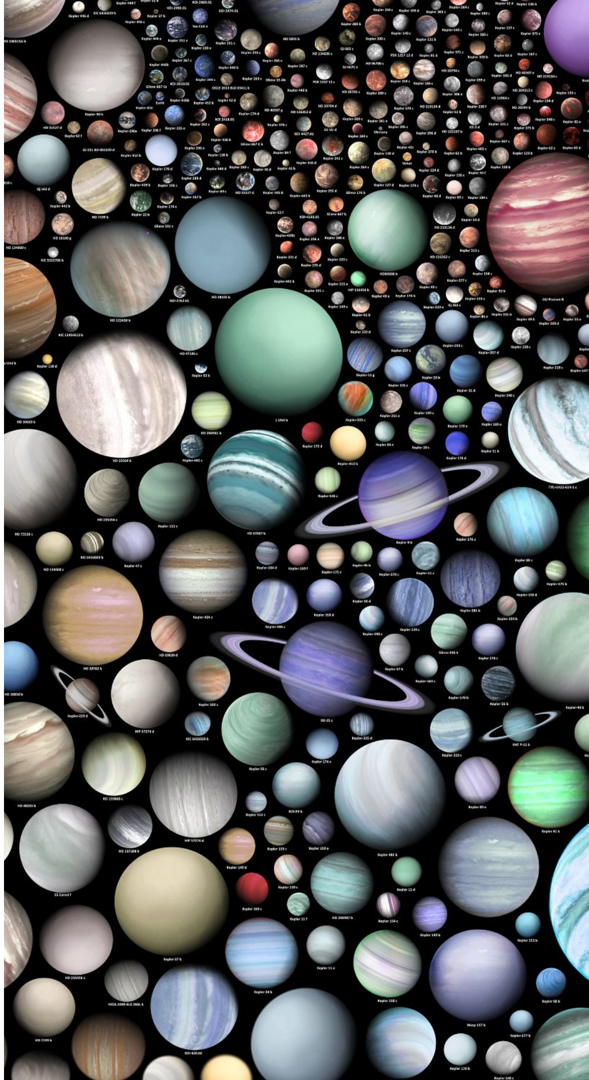
How can we find exoplanets?

Exoplanet = planet orbiting a star other than the sun

Only in science fiction until 1992!

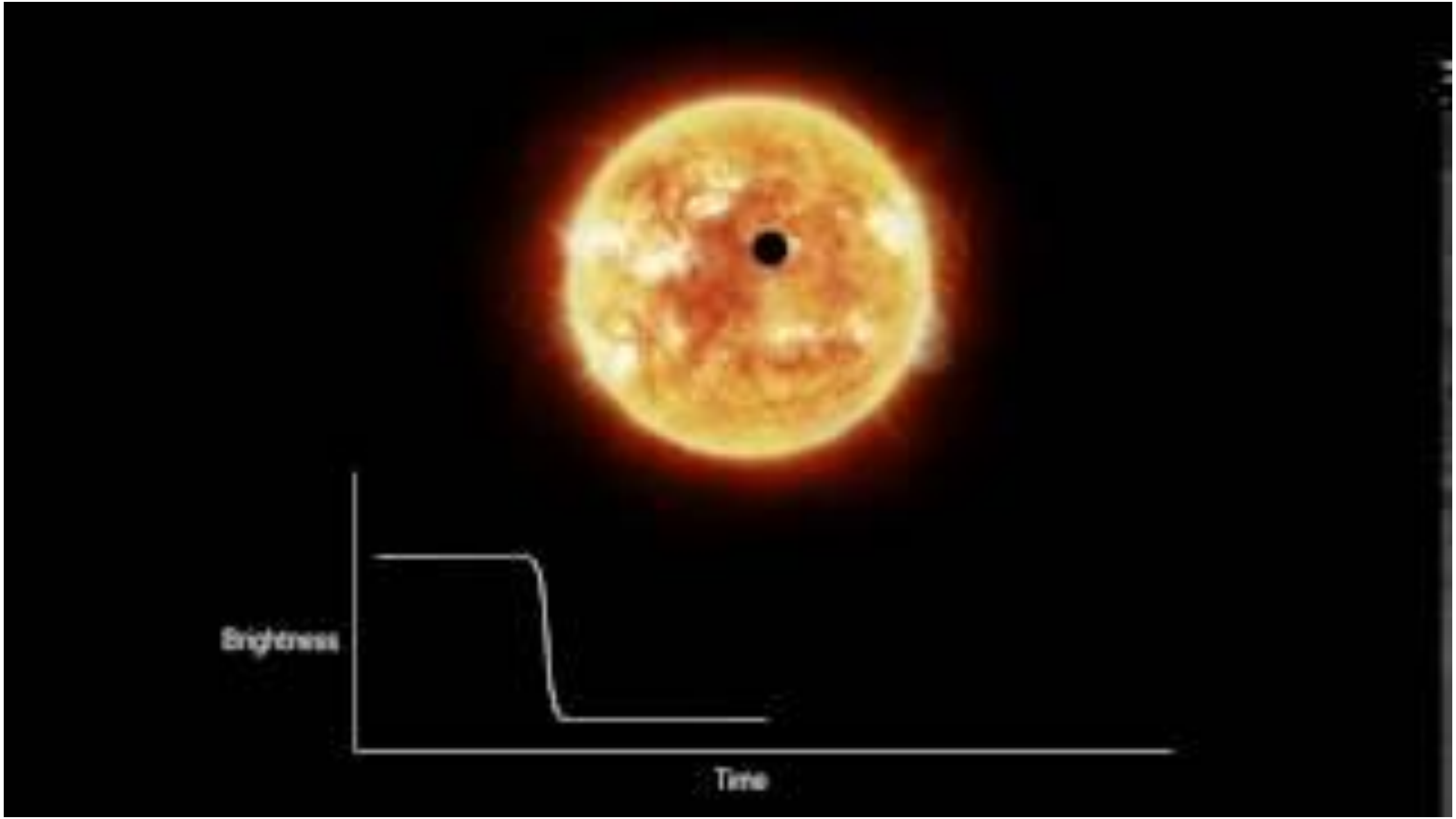
- Since then, thousands discovered

Discovery Method	Number of Planets
Astrometry	2
Imaging	67
Radial Velocity	1048
Transit	4092
Transit timing variations	25
Eclipse timing variations	17
Microlensing	200
Pulsar timing variations	7
Pulsation timing variations	2
Orbital brightness modulations	9
Disk Kinematics	1



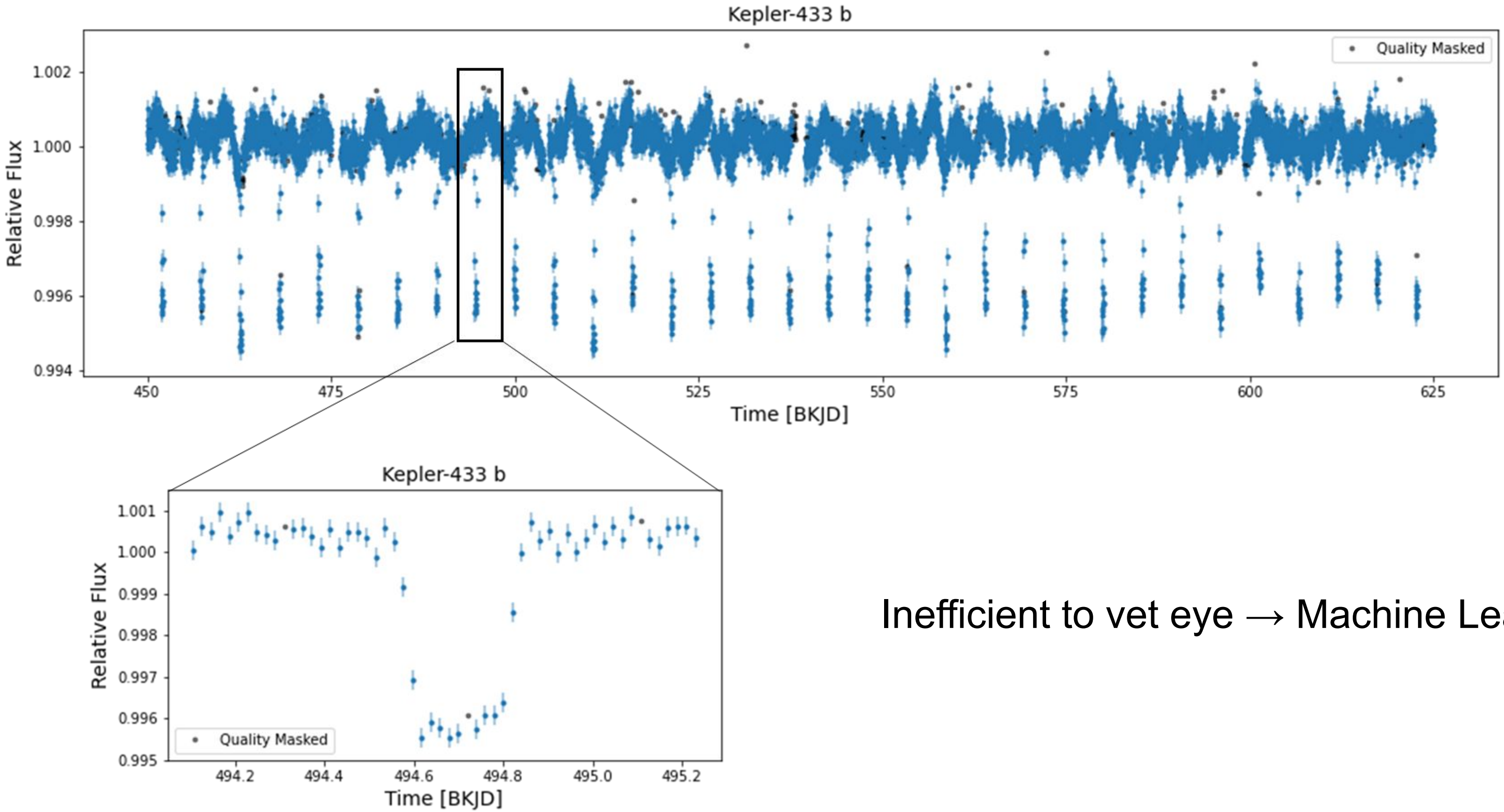
Graphic: Martin Vargic,Table: NExSci

What are transits?



Animation: ESA

How can we use transits to detect new planets?



Inefficient to vet eye → Machine Learning

What has already been done?

- Early 2000s: **machine-aided** approaches
  - Box-Fitting-Least-Squares (e.g. Kovacs et al., 2002)
  - Bayesian-based analysis of transit likelihood (Aigrain and Favata, 2002)
- Mid-2010s: First widespread use of **ML** for transit detection (DTs, RFs, SVMs, KNNs)
  - Time and frequency domain data
  - Mostly used for “vetting” of “pre-triaged” observations.
- Late 2010s: Application of **Deep Learning** algorithms
  - Largely for vetting
  - 1D-convolutional networks (e.g. Zucker and Giryes, 2018)
  - Use with phase-folded curves (e.g. Pearson et al., 2018)

Our Task: ML classification of lightcurves without previous vetting.

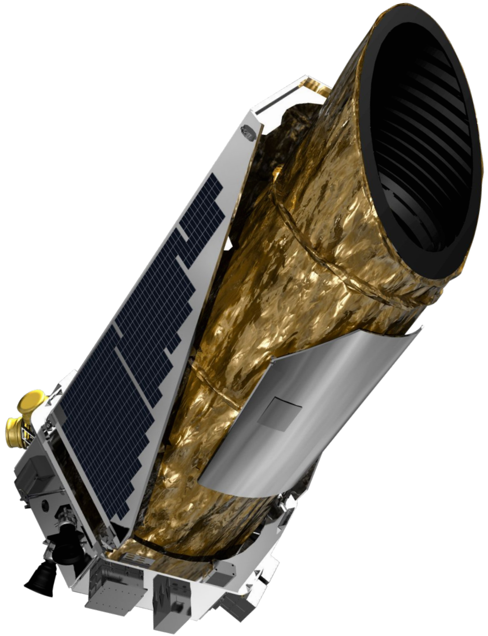
Data

Kepler mission is one of the largest exoplanet surveys (2708 confirmed detections)

- High exposure time → able to observe dimmer, more distant targets
- Prioritized Main Sequence stars for which Earth-like planets would be detectable
- Thousands of lightcurves of stellar targets, some with transiting exoplanets
- Lightcurves are publicly available online
- NASA Exoplanet Archive for class labels

Lightcurve data:

- Several 90-day quarters, at a cadence of 30min
- Flux (light intensity) as a function of time from each target star
- Intervals of missing data due to telescope operation, and many quality flagged datapoints.

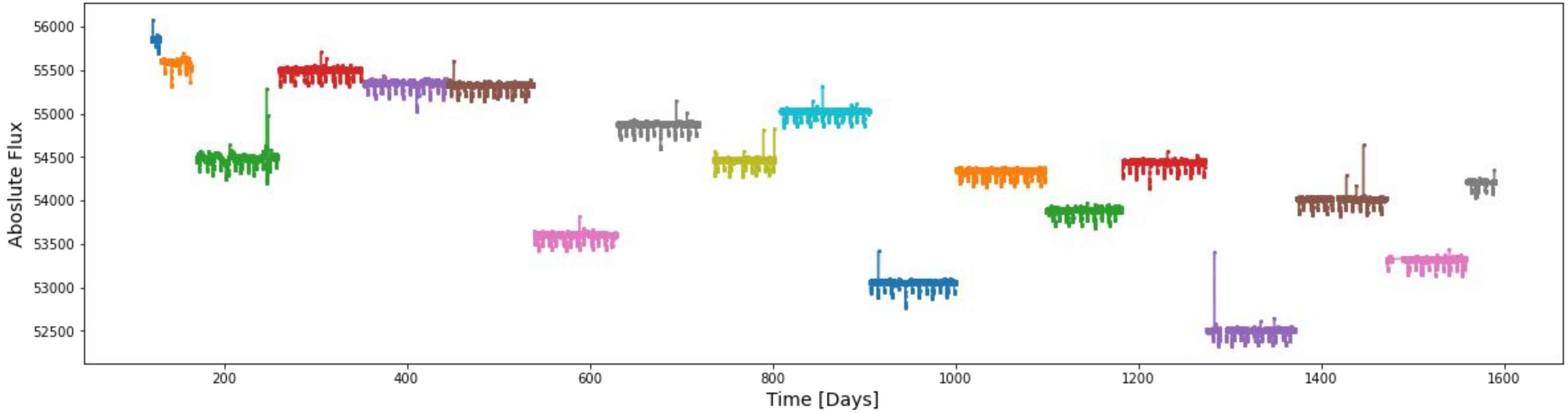


The Kepler satellite

Preprocessing

Preprocessing steps:

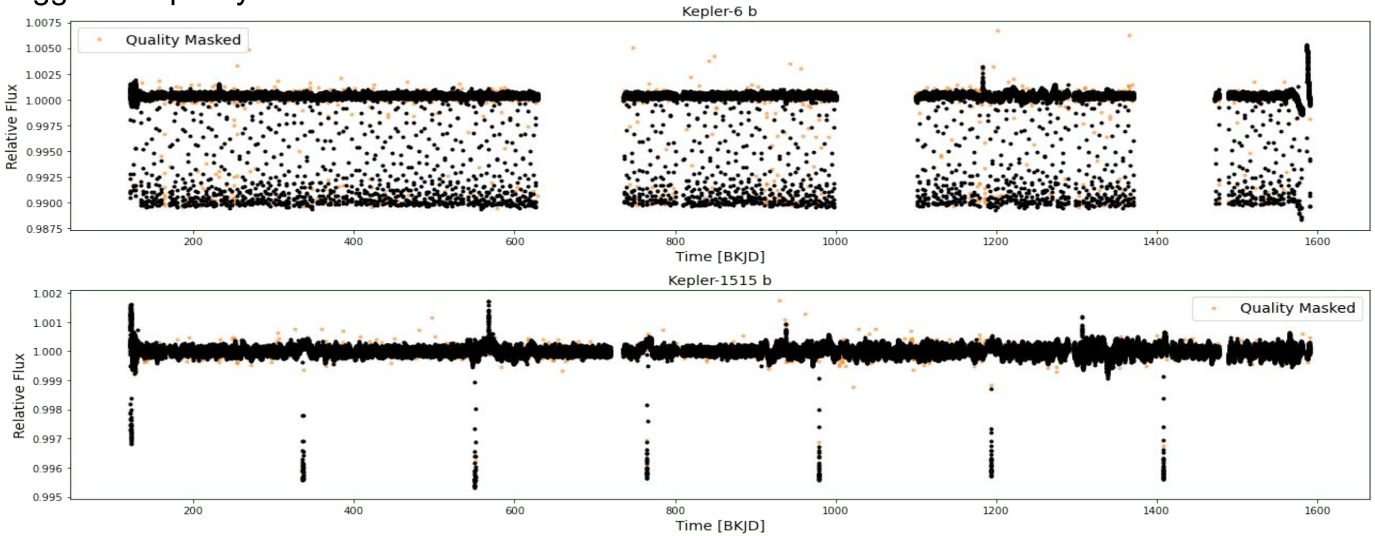
1. Fit and **remove** quarterly linear trends
2. **Stitch** quarters



Preprocessing

Preprocessing steps:

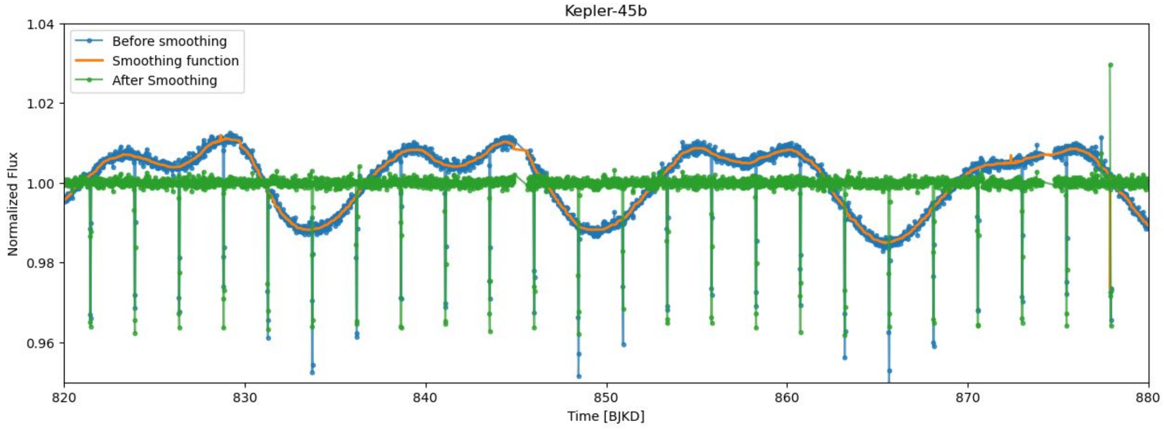
1. Fit and **remove** quarterly linear trends
2. **Stitch** quarters
3. Mask data flagged for quality issues



Preprocessing

Preprocessing steps:

1. Fit and **remove** quarterly linear trends
2. **Stitch** quarters
3. **Mask data** flagged for quality issues
4. Remove stellar variability by **smoothing**



Preprocessing

Preprocessing steps:

1. Fit and **remove** quarterly linear trends
2. **Stitch** quarters
3. **Mask data** flagged for quality issues
4. Remove stellar variability by **smoothing**
5. Fill in missing values by **interpolation**

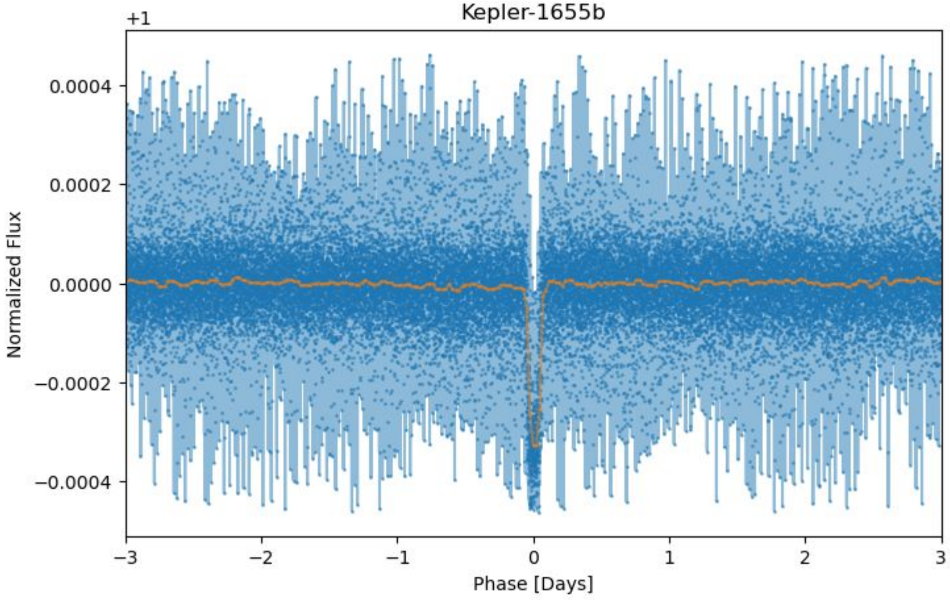
Feature Engineering

First, we ran our classification algorithms on **“raw” lightcurves**

- Features → flux values at each timestep

But, **feature comparison** between lightcurves this way is not meaningful

- Many algorithms compare “distance” or create decision boundaries based on the same feature for each observation.
- Solution: **phase folding**
    - Create boxed-least-squares periodogram
    - Fold on period of maximum power
    - Bin to reduce noise and construct equal length observations



Also explored **Recursive feature selection (RFE)** and **SelectKBest**

- feature selection not very meaningful for our data

Model selection

For the selection of model we considered the following classification metrics:

- **Accuracy**: ratio of correctly classified instances to the total number of instances.
- **Recall:** ratio of true positives to the sum of true positives and false positives
- **Precision:** ratio of true positives to the sum of true positives and false negative
- **F1-score:** harmonic mean of recall and precision

Among the 4 we decided to evaluate all the models based on the F1-score.

$$(10.1)\quad Accuracy = \frac{T_p + T_n}{T_p + T_n + F_p + F_n}$$

$$(10.2)\quad Precision = \frac{T_p}{T_p + F_p}$$

$$(10.3)\quad Recall = \frac{T_p}{T_p + T_n}$$

$$(10.4)\quad F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Algorithms: KNN

Motivation: baseline to compare to other classification methods.

The **KNN** algorithm:

1. **Store training data** as points in feature space.
2. Find the k **nearest neighbors** to each new observation using a pre-defined distance metric (we use Euclidean)
   - Because of the nature of our observations, future work could define a more meaningful metric
3. Take the **mode of the labels** of the k neighbors to predict the label of the new

KNN is considered a **“lazy learner”** because training involves merely storing the training data

- Does not produce a true “model” that one can examine to gain insight into our data

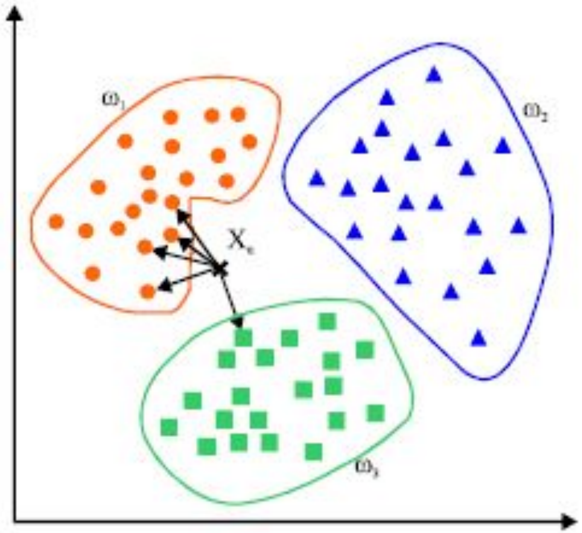


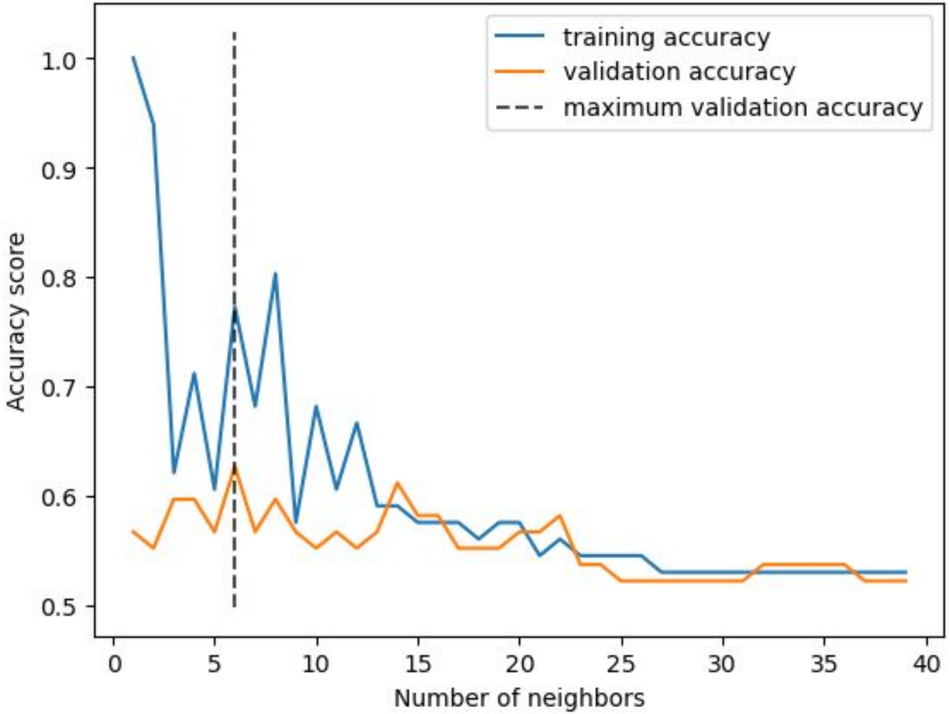
Image: medium.com

Algorithms: KNN

Results on **Non-Engineered Features**

Tuning hyperparameter k:

- Best validation results for k=6



Algorithms: KNN

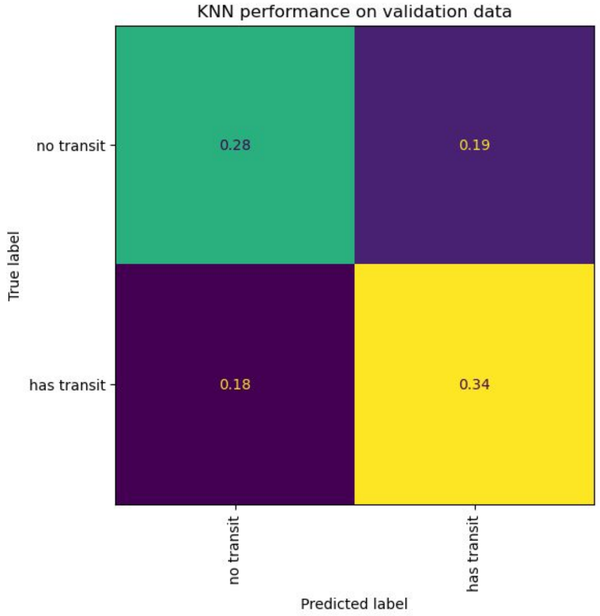
Results on **Non-Engineered Features**

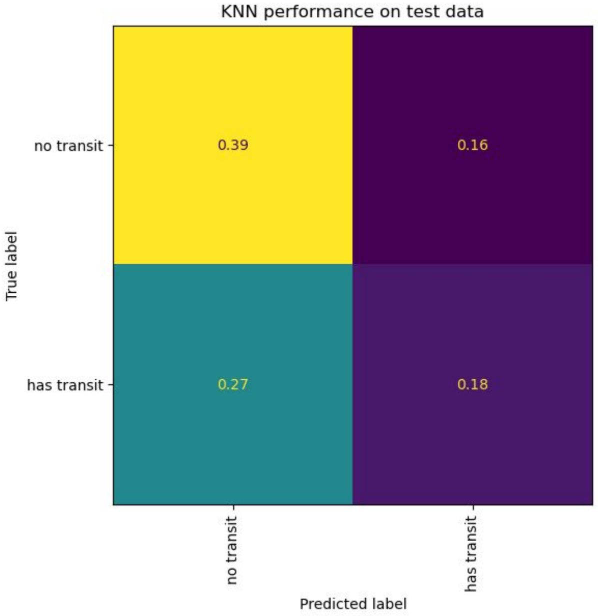
Tuning hyperparameter k:

- Best validation results for k=6

Performance:

```
Validation Data:
    Accuracy: 0.627
    Precision: 0.639
    Recall: 0.657
    F1: 0.648
Test Data:
    Accuracy: 0.567
    Precision: 0.522
    Recall: 0.4
    F1: 0.453
```



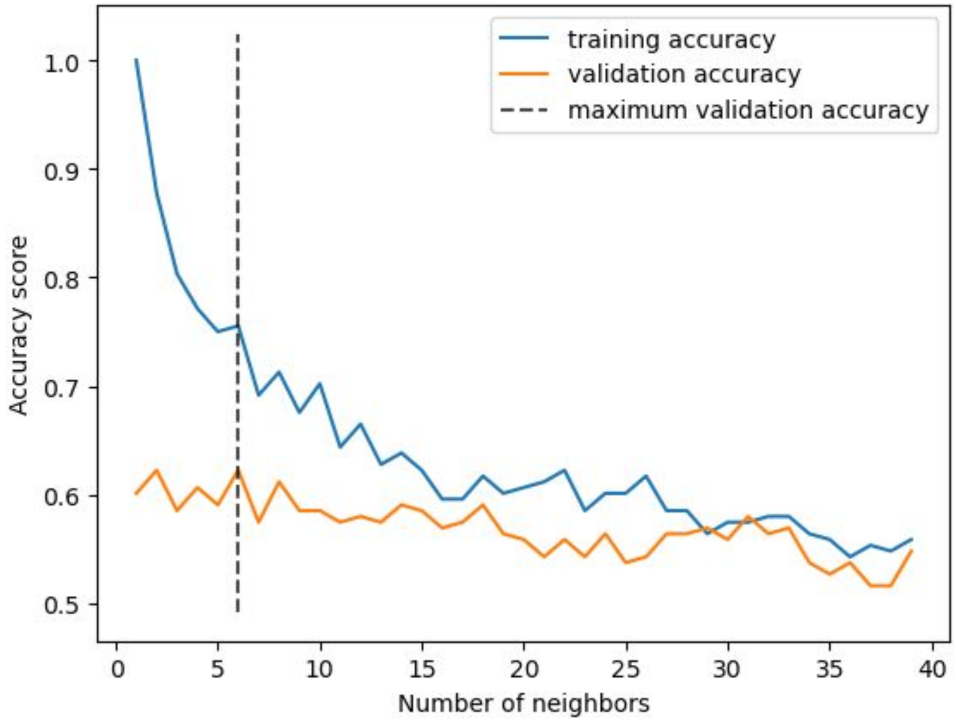


Algorithms: KNN

Results on **Engineered Features**

Tuning hyperparameter k:

- Best validation results for k=6



Algorithms: KNN

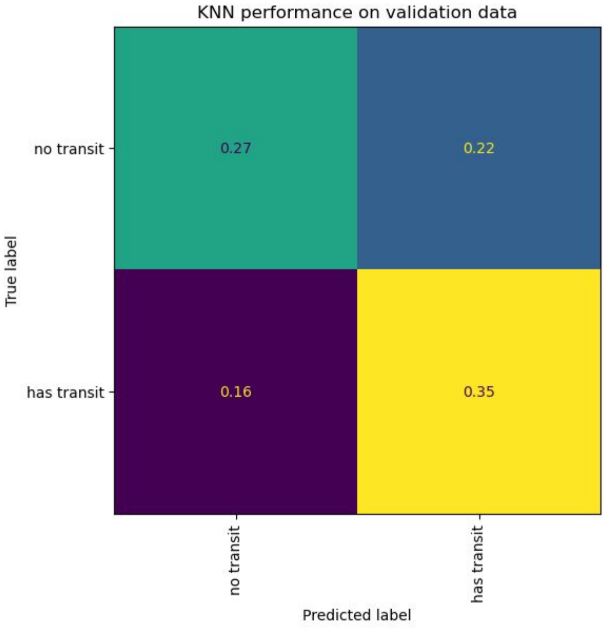
Results on **Engineered Features**

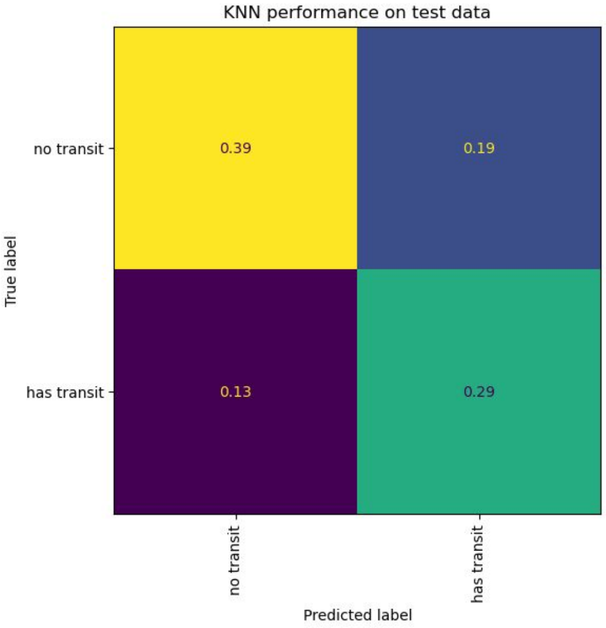
Tuning hyperparameter k:

- Best validation results for k=6

Performance:

```
Validation Data:
    Accuracy: 0.622
    Precision: 0.617
    Recall: 0.688
    F1: 0.65
Test Data:
    Accuracy: 0.683
    Precision: 0.604
    Recall: 0.696
    F1: 0.647
```





Algorithms: Random Forest

Motivation: less prone to overfitting, good for high-dimensional data

The **Random Forest** algorithm**:** collection of Decision Tree (DT) classifiers, each trained on subsamples of the data

1. For each DT
    a. Construct a **subset** of training data
    b. Iteratively **determine splits** at each node based on some metric
        - We choose information gain
    c. Stop once all nodes contain only **one class**
    d. **Prune** to reduce overfitting
        - We set a maximum tree depth, but we still overfit
    e. Label new observations with the **mode of training labels** in the reached leaf node
2. Classification is assigned using the **majority vote**.

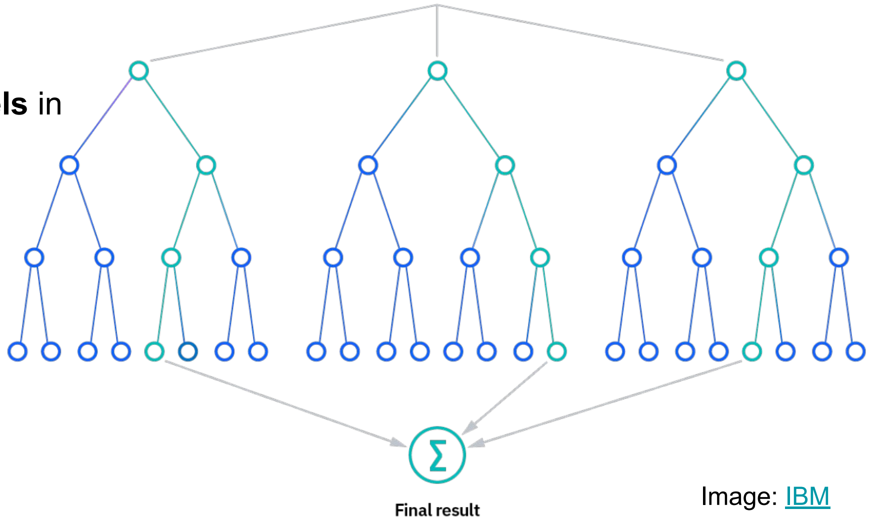


Image: IBM

Initially, find a bias to misclassify observations as negative rather than as positive.

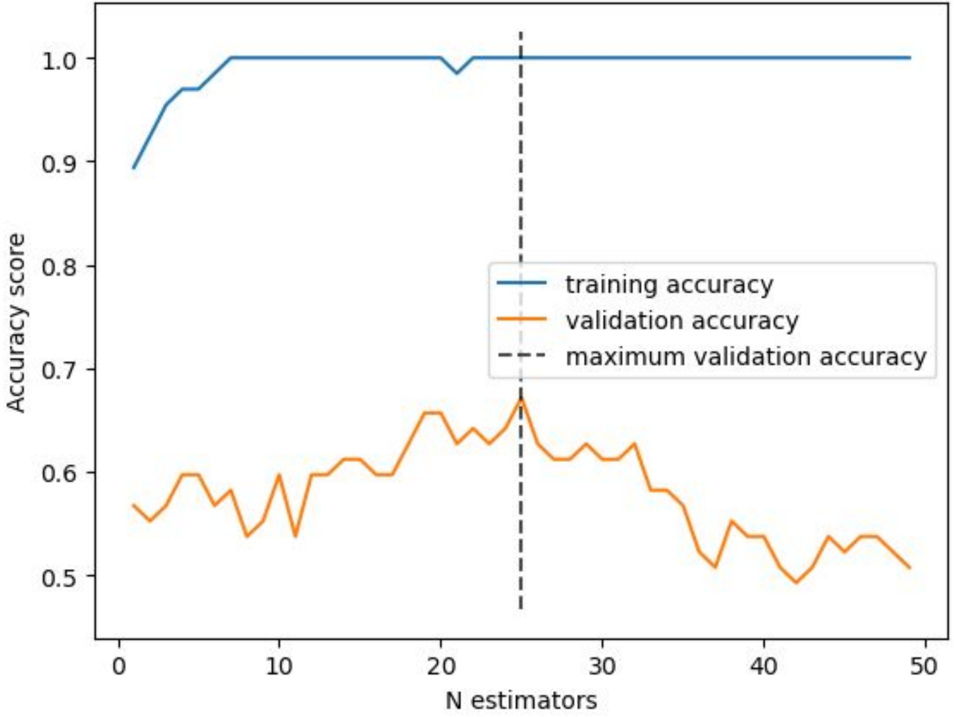
- Based on science case, prefer the opposite
- Add a **set of class weights** to encourage positive classifications

Algorithms: Random Forest

Results on **Non-Engineered Features**

Tuning hyperparameter n_estimators (number of classifiers):

- Best validation results for n_estimators = 25



Algorithms: Random Forest

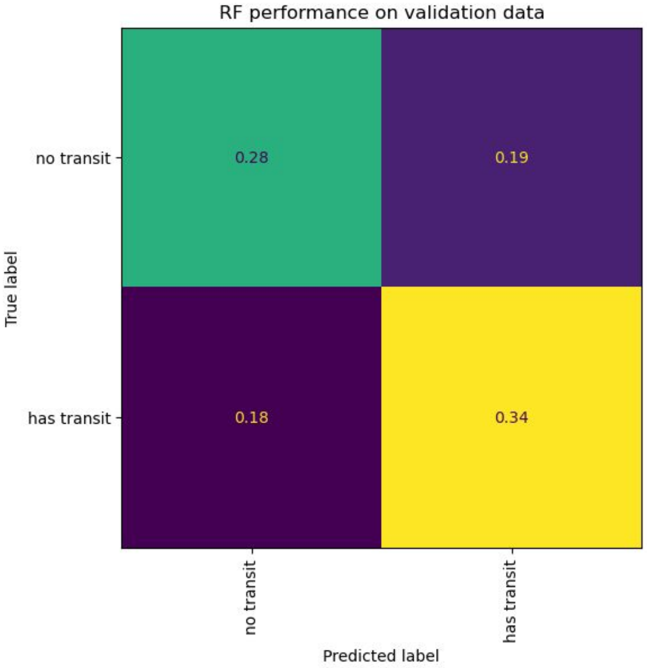
Results on **Non-Engineered Features**

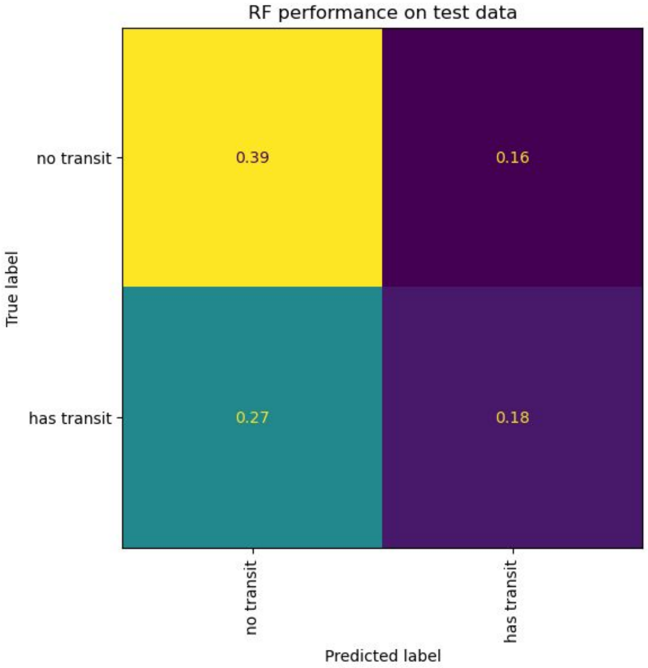
Tuning hyperparameter n_estimators (number of classifiers):

- Best validation results for n_estimators = 25

Performance:

```
Validation Data:
    Accuracy: 0.672
    Precision: 0.651
    Recall: 0.8
    F1: 0.718
Test Data:
    Accuracy: 0.627
    Precision: 0.568
    Recall: 0.7
    F1: 0.627
```



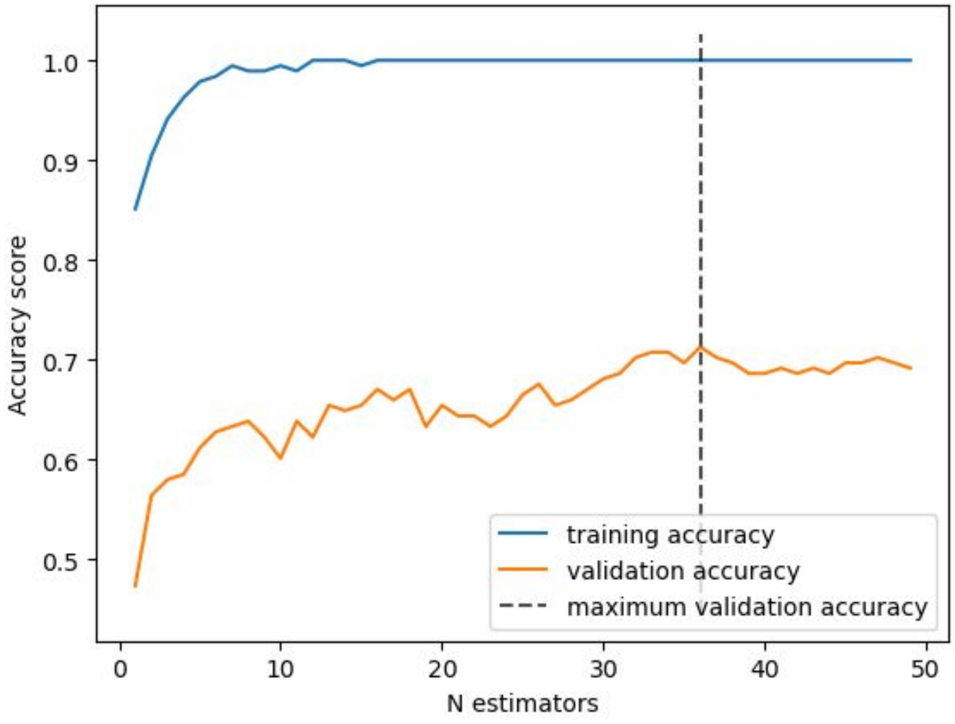


Algorithms: Random Forest

Results on **Engineered Features**

Tuning hyperparameter n_estimators (number of classifiers):

- Best validation results for n_estimators = 36



Algorithms: Random Forest

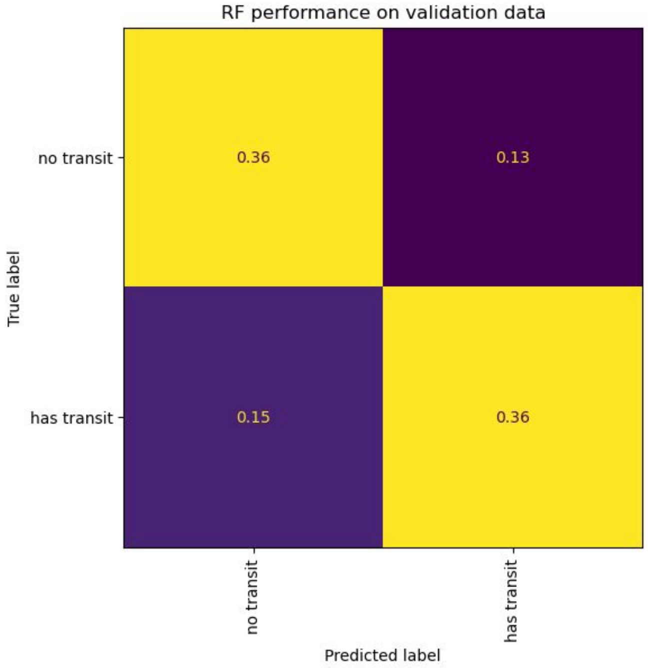
Results on **Engineered Features**

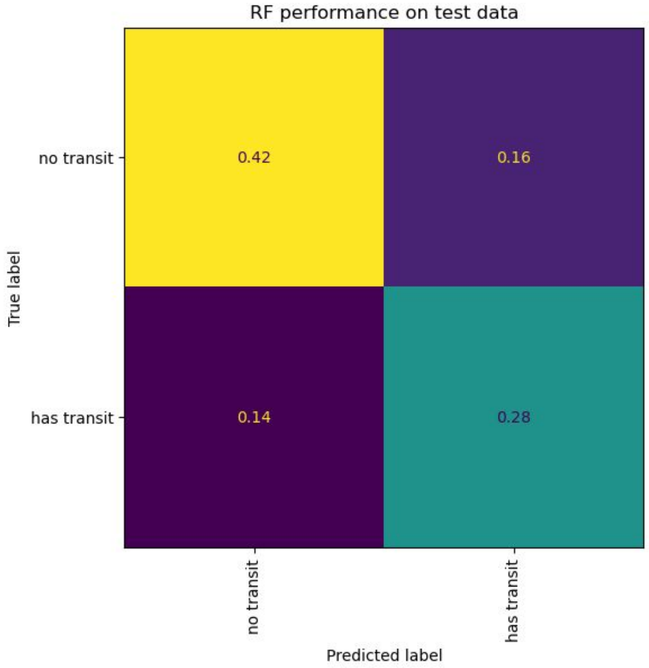
Tuning hyperparameter n_estimators (number of classifiers):

- Best validation results for n_estimators = 36

Performance:

```
Validation Data:
    Accuracy: 0.713
    Precision: 0.728
    Recall: 0.698
    F1: 0.713
Test Data:
    Accuracy: 0.698
    Precision: 0.631
    Recall: 0.671
    F1: 0.65
```





Algorithms: Weighted Logistic Regression

The **Logistic Regression** algorithm:

1. Construct a decision boundary in feature space by finding the logistic function that best splits the data by class label.
   a. Best-split is determined by minimizing the log-loss of the predicted and true values
   b. Can return likelihoods; we use a classification threshold of 0.5.

We were **unable to achieve good performance**

- Model either classified all observations as positive or all observations as negative each time
  - Can't closely fit our decision boundary
- Add class weights to encourage the model to make both positive and negative classifications
  - Highly sensitive to class weights near critical value

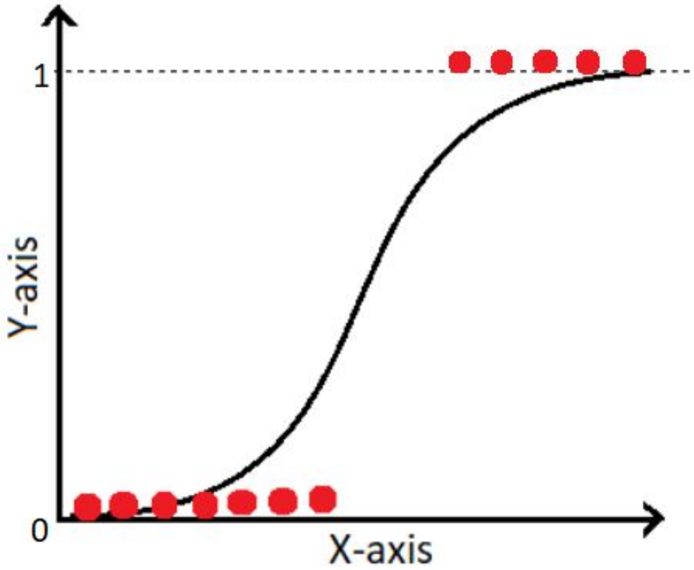


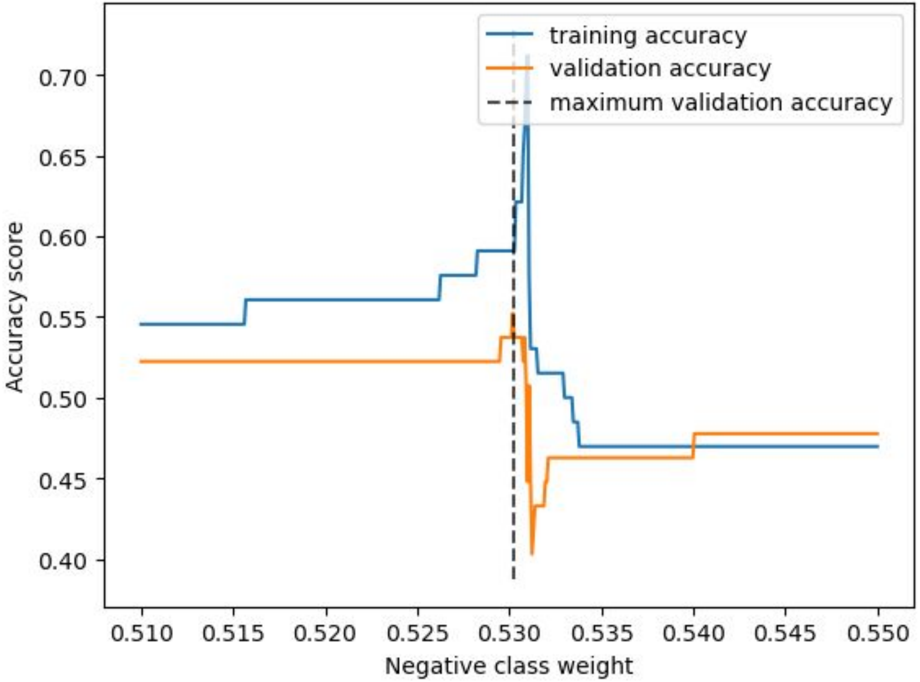
Image: medium.com

Algorithms: Logistic Regression

Results on **Non-Engineered Features**

Tuning hyperparameter class weights (positive weight = 1 - negative weight):

- Best validation results for negative weight = 0.5302004



Algorithms: Logistic Regression

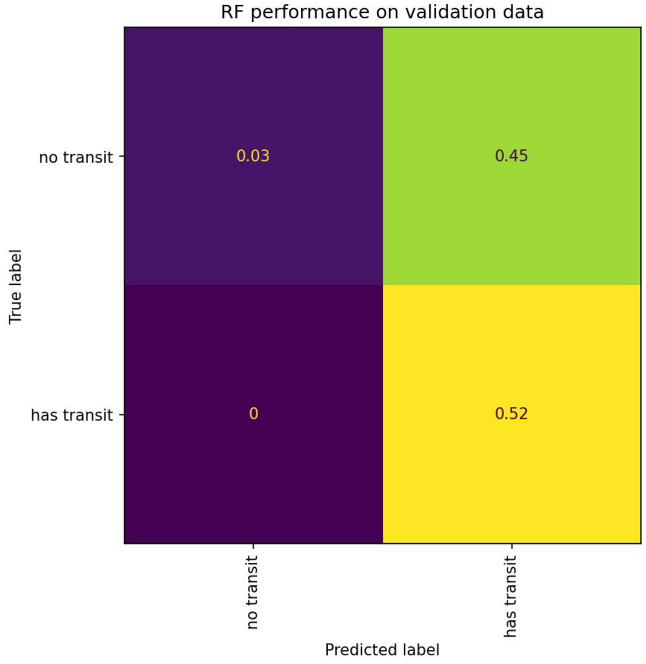
Results on **Non-Engineered Features**

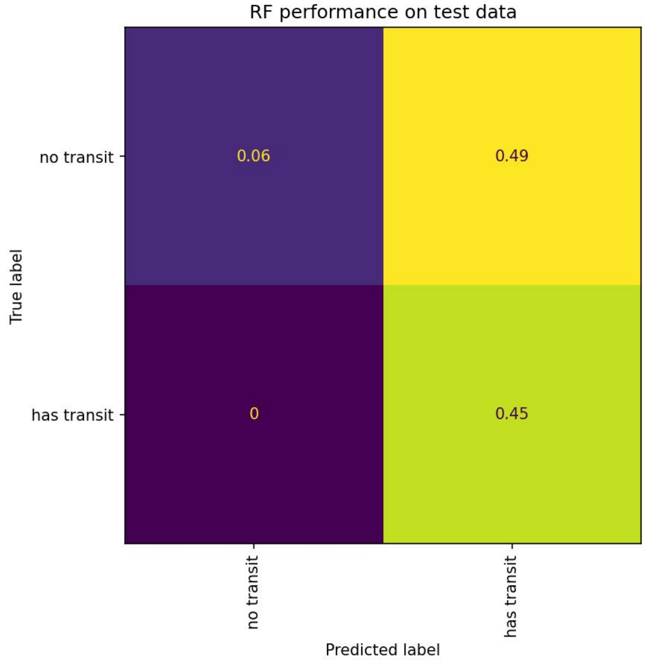
Tuning hyperparameter class weights (positive weight = 1 - negative weight):

- Best validation results for negative weight = 0.5302004

Performance:

```
Validation Data:
   Accuracy: 0.552
   Precision: 0.538
   Recall: 1.0
   F1: 0.7
Test Data:
   Accuracy: 0.507
   Precision: 0.476
   Recall: 1.0
   F1: 0.645
```



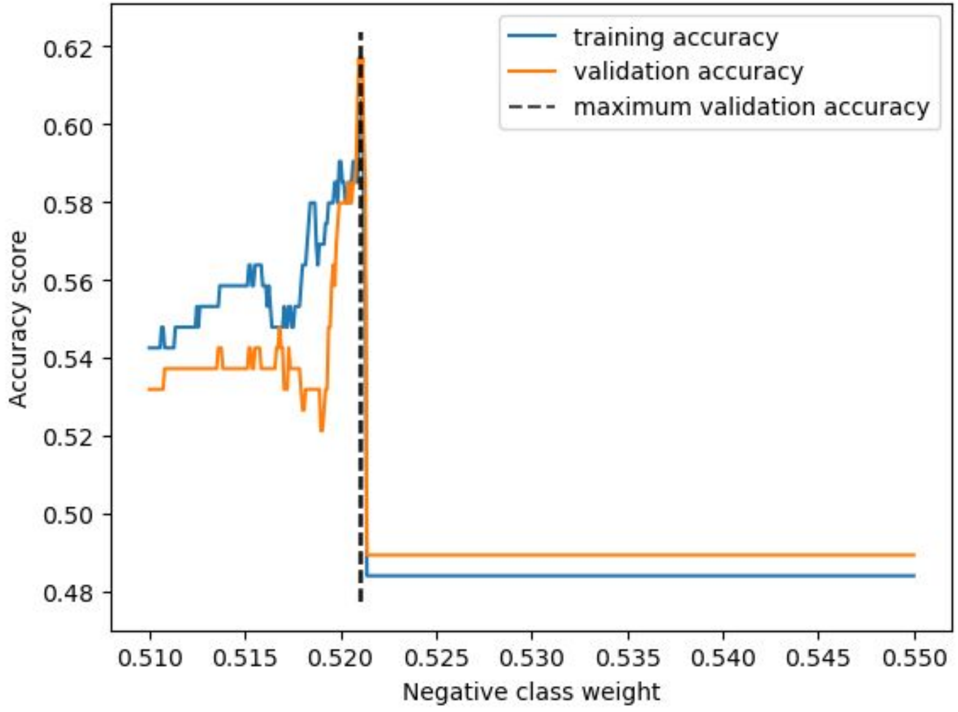


Algorithms: Logistic Regression

Results on **Engineered Features**

Tuning hyperparameter class weights (positive weight = 1 - negative weight):

- Best validation results for negative weight = 0.52098196



Algorithms: Logistic Regression

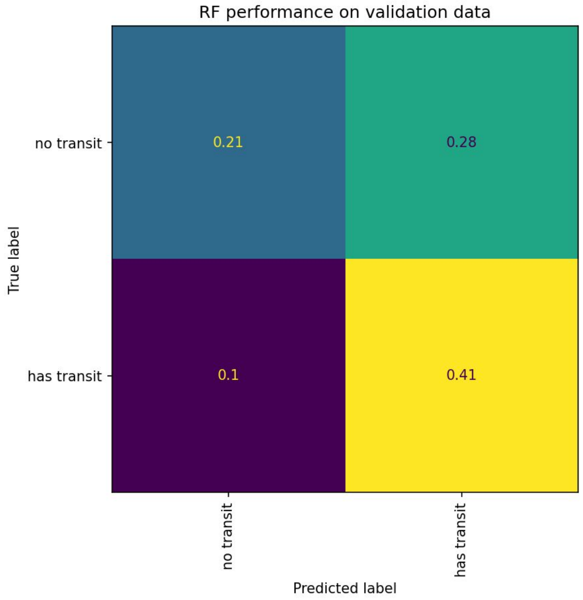
Results on **Engineered Features**

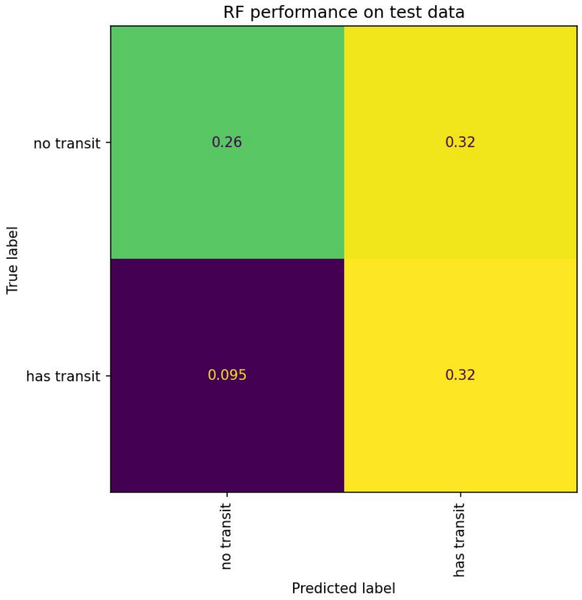
Tuning hyperparameter class weights (positive weight = 1 - negative weight):

- Best validation results for negative weight = 0.52098196

Performance:

```
Validation Data:
   Accuracy: 0.617
   Precision: 0.592
   Recall: 0.802
   F1: 0.681
Test Data:
   Accuracy: 0.587
   Precision: 0.504
   Recall: 0.772
   F1: 0.61
```





Algorithms: SVM

Motivation: SVMs are efficient in managing high-dimensional data and identifying non-linear relationships.

The **SVM** Algorithm: find the optimal hyperplane in an N-dimensional space that can separate the data points in different classes

1. Identify the **support vectors**, which are the data points closed to the decision hyperplane.
2. **Maximize** the distance between the support vectors and the hyperplane while minimizing the classification errors.
3. In case of non-linear kernel, the input data is implicitly **transformed** into higher dimensional feature space by the kernel function.

The final prediction is based on the decision function that is calculated using the weights (hyperplane equation) learned during training and a bias term. If the sign of the decision function value is positive then predict the positive label otherwise the negative label.

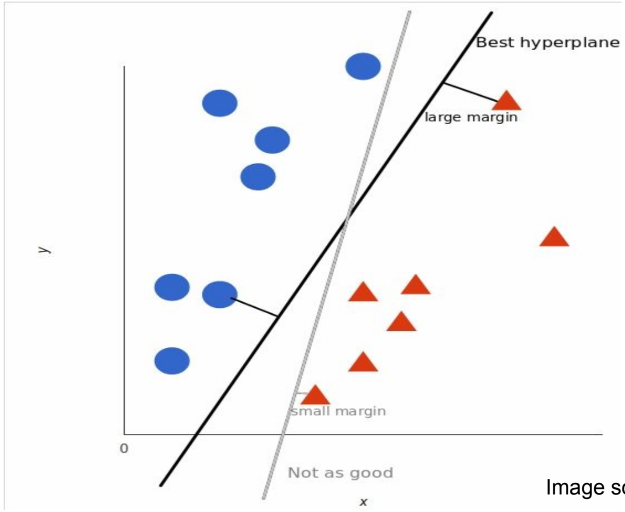


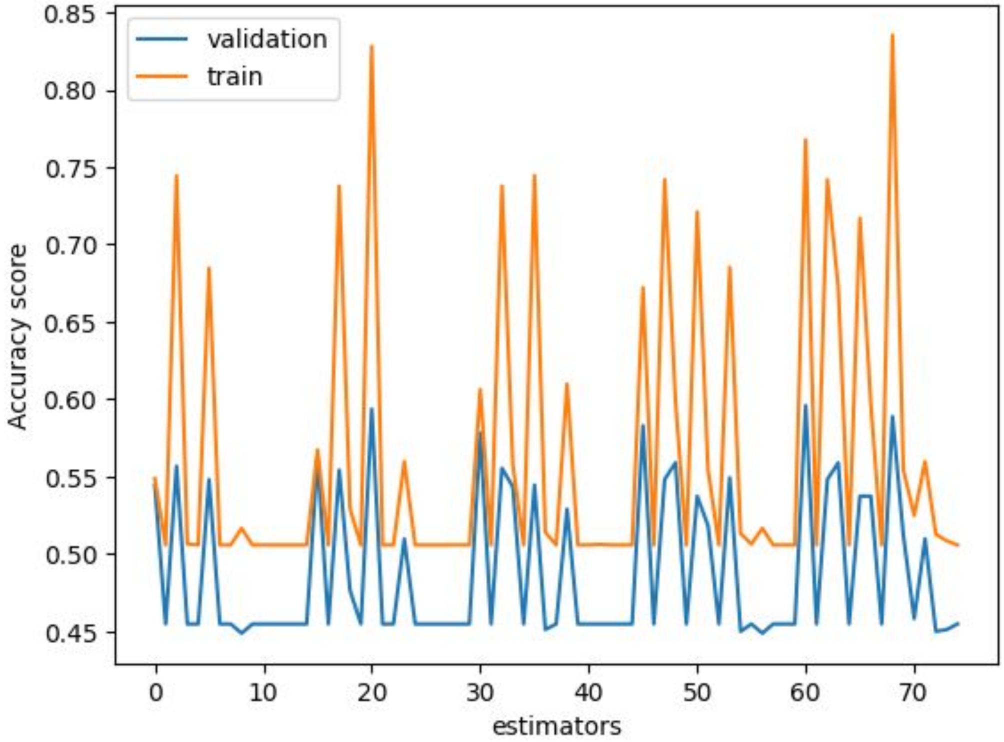
Image source: Monkey Learn

Algorithms: SVM

Results on **Non-Engineered Features**

Tuning hyperparameter C (regularization param), gamma (Kernel coefficient), kernel:

- Best validation results for C=1000, gamma=1, and kernel=rbf (radial basis function)



Algorithms: SVM

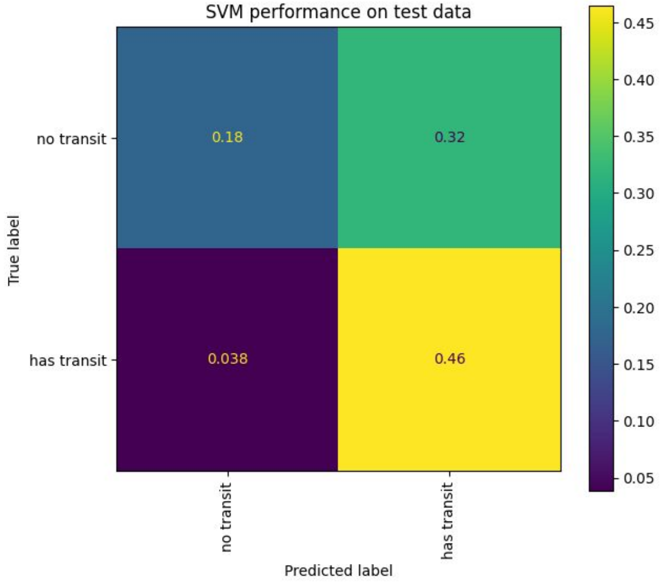
Results on **Non-Engineered Features**

Tuning hyperparameter C (regularization param), gamma (Kernel coefficient), kernel:

- Best validation results for C=1000, gamma=1, and kernel=rbf (radial basis function)

Performance:

```
Validation Data:
        Accuracy 0.767
        Precision 0.681
        Recall 1.0
        F1 0.811
Test Data:
        Accuracy 0.641
        Precision 0.591
        Recall 0.924
        F1 0.721
```

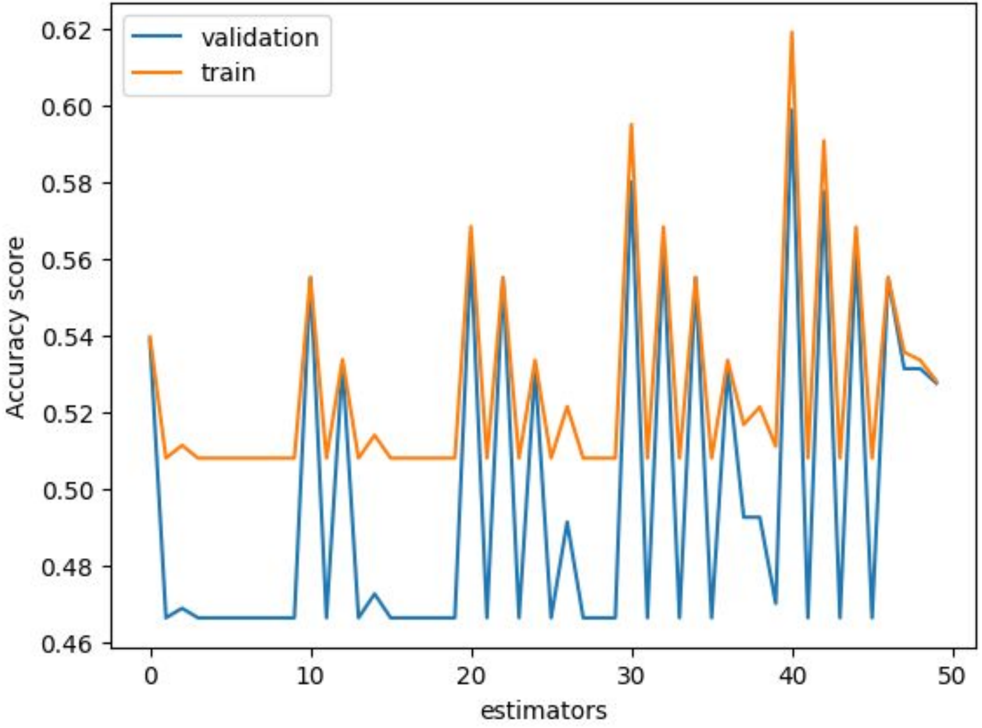


Algorithms: SVM

Results on **Engineered Features**

Tuning hyperparameter C (regularization param), gamma (Kernel coefficient), kernel:

- Best validation results for C=1000, gamma=1, and kernel=rbf (radial basis function)



Algorithms: SVM

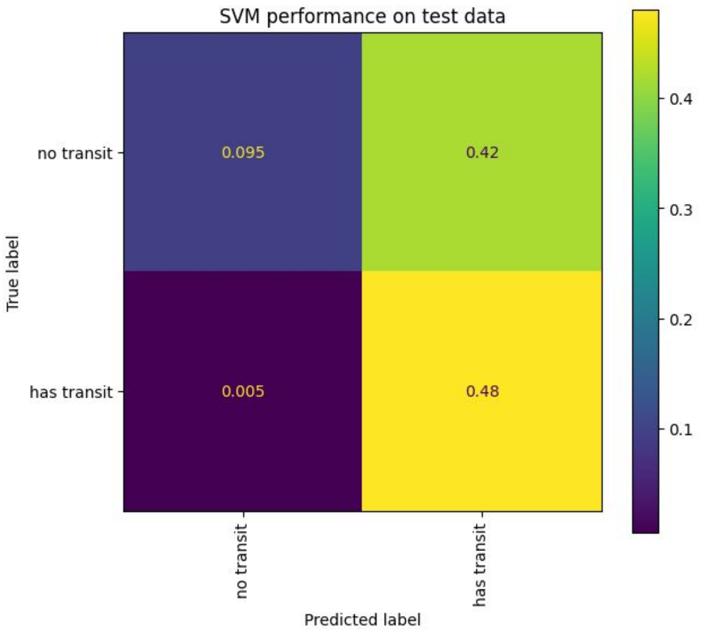
Results on **Engineered Features**

Tuning hyperparameter C (regularization param), gamma (Kernel coefficient), kernel:

- Best validation results for C=1000, gamma=1, and kernel=rbf (radial basis function)

Performance:

```
Validation Data:
        Accuracy 0.618
        Precision 0.57
        Recall 0.993
        F1 0.724
Test Data:
        Accuracy 0.575
        Precision 0.533
        Recall 0.99
        F1 0.693
```



Algorithms: AdaBoost

The **AdaBoost** Algorithm: collection of Decision Tree (DT) classifiers, improving on each iteration through weighted voting

1. Initialization of **weights**. Normally consider 1/n_samples
2. **Train** an estimator using current weights
3. Calculate the **error_rate** based on the prediction of the estimator
4. **Update** the weights using the learning rate and error_rate.
5. **Repeat** the steps 2-4 for n estimators

Final prediction is assigned as the **weighted sum** of all the DT after all the iterations are done.

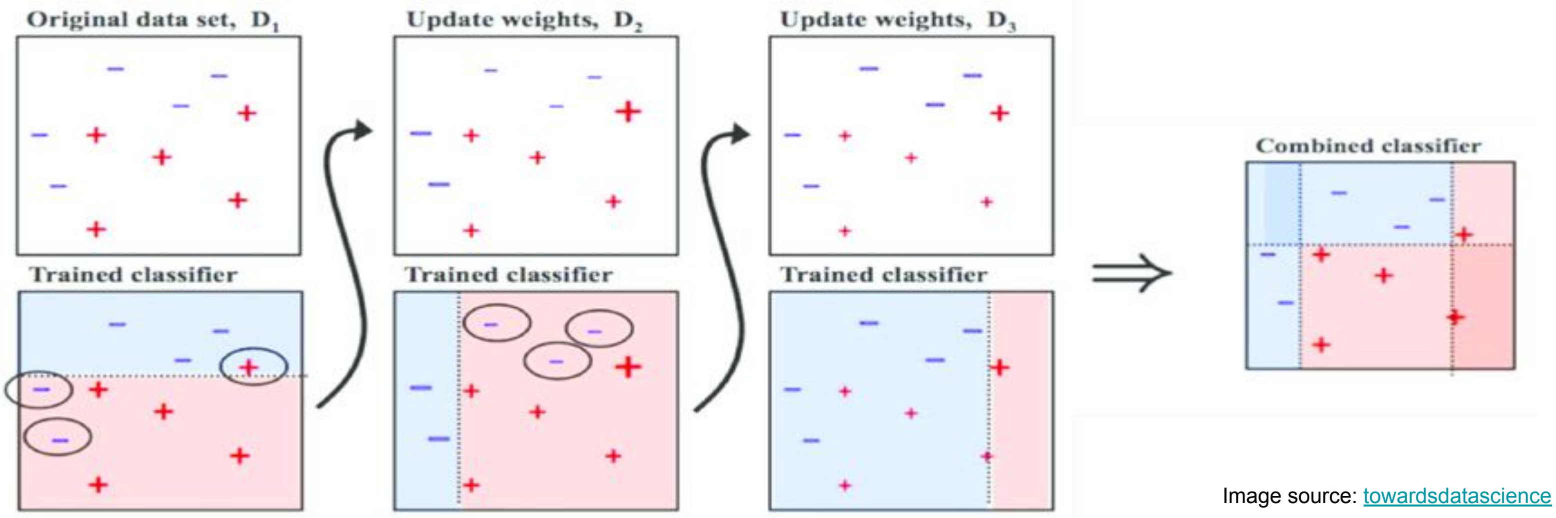


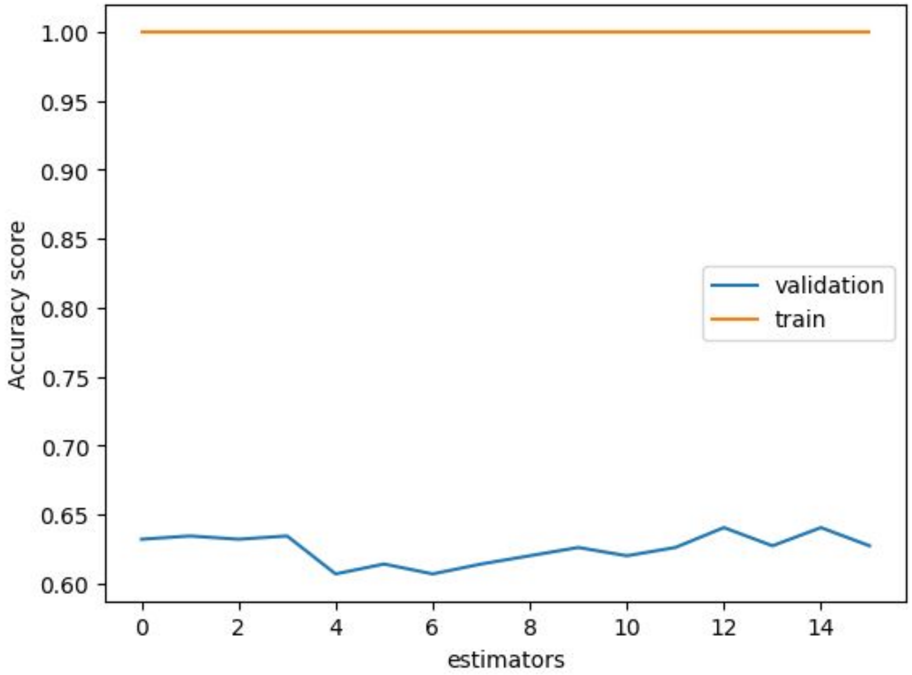
Image source: towardsdatascience

Algorithms: AdaBoost

Results on **Non-Engineered Features**

Tuning hyperparameter max_depth and min_samples_leaf for base estimator along with learning_rate and n_estimators:

- Best validation results for base_estimator__max_depth=20, base_estimator__min_samples_leaf=7, learning_rate=0.01, n_estimators=60



Algorithms: AdaBoost

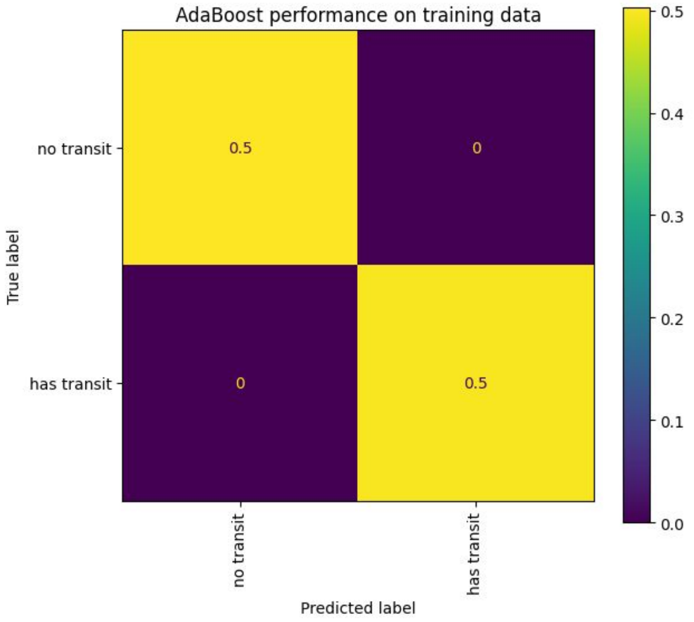
Results on **Non-Engineered Features**

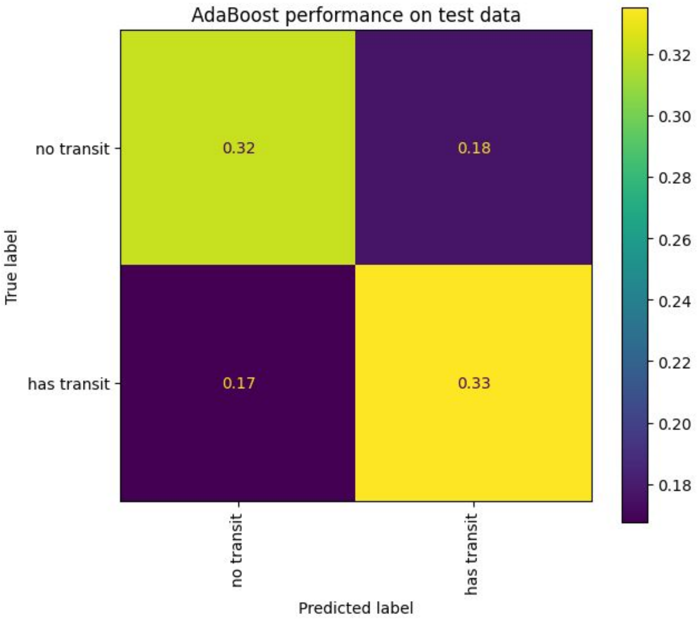
Tuning hyperparameter max_depth and min_samples_leaf for base estimator along with learning_rate and n_estimators:

- Best validation results for base_estimator__max_depth=20, base_estimator__min_samples_leaf=7, learning_rate=0.01, n_estimators=60

Performance:

```
Validation Data:
        Accuracy 1.0
        Precision 1.0
        Recall 1.0
        F1 1.0
Test Data:
        Accuracy 0.656
        Precision 0.654
        Recall 0.667
        F1 0.66
```



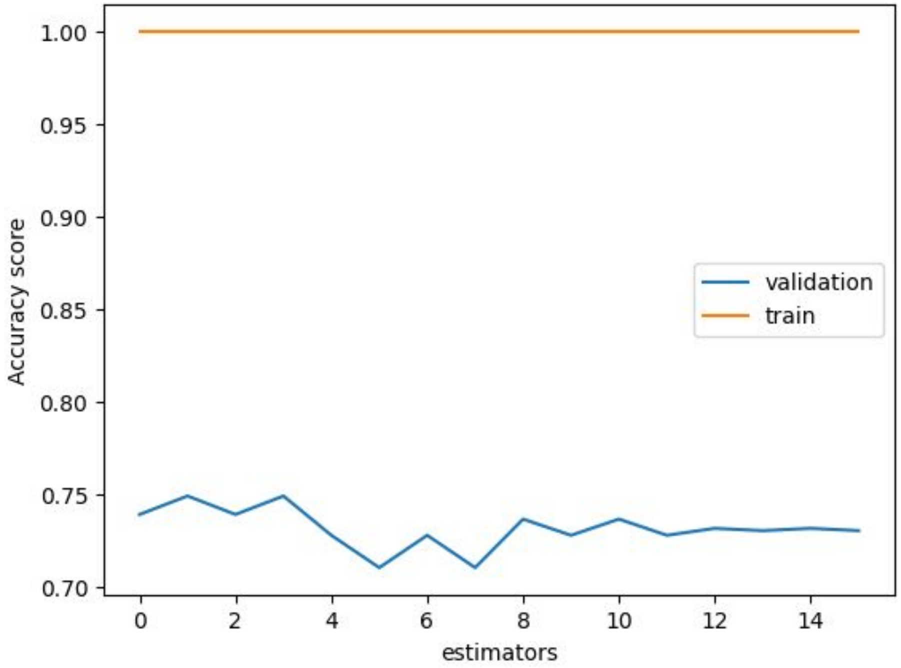


Algorithms: AdaBoost

Results on **Engineered Features**

Tuning hyperparameter max_depth and min_samples_leaf for base estimator along with learning_rate and n_estimators:

- Best validation results for base_estimator__max_depth=20, base_estimator__min_samples_leaf=7, learning_rate=0.01, n_estimators=60



Algorithms: AdaBoost

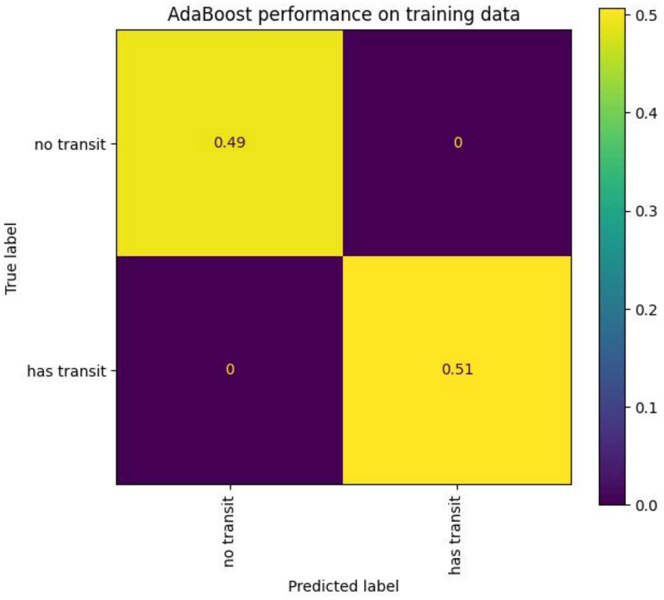
Results on **Engineered Features**

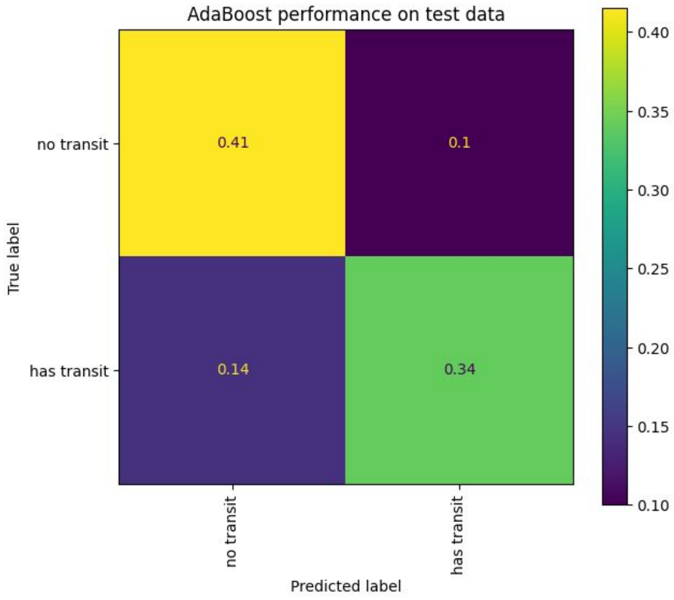
Tuning hyperparameter max_depth and min_samples_leaf for base estimator along with learning_rate and n_estimators:

- Best validation results for base_estimator__max_depth=10, base_estimator__min_samples_leaf=5, learning_rate=0.01, n_estimators=70

Performance:

```
Validation Data:
        Accuracy 1.0
        Precision 1.0
        Recall 1.0
        F1 1.0
Test Data:
        Accuracy 0.755
        Precision 0.773
        Recall 0.701
        F1 0.735
```





Algorithms: Gradient Boost

The **Gradient Boost** Algorithm: collection of Decision Tree (DT) classifiers, improving on each iteration by correcting the errors of its predecessors

1. Initialization: create a simple model of DT as the first weak learner.
2. Calculate the **error_rate/residual** based on the prediction of the estimator
3. **Iteratively build** weak learners by reducing the residual error, and correcting the mistakes of the previous model. This is controlled by the learning rate.
4. Next the ensemble prediction is updated as the weighted sum of predictions from new learner and the previous ones.
5. **Repeat** the steps 2-4 for n estimators

Final prediction is assigned as the **weighted sum** of all the DT after all the iterations are done.

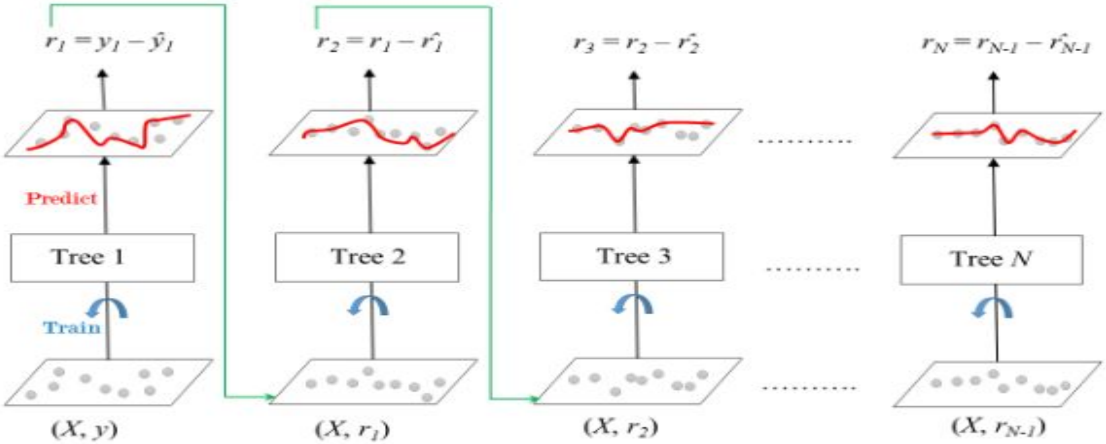


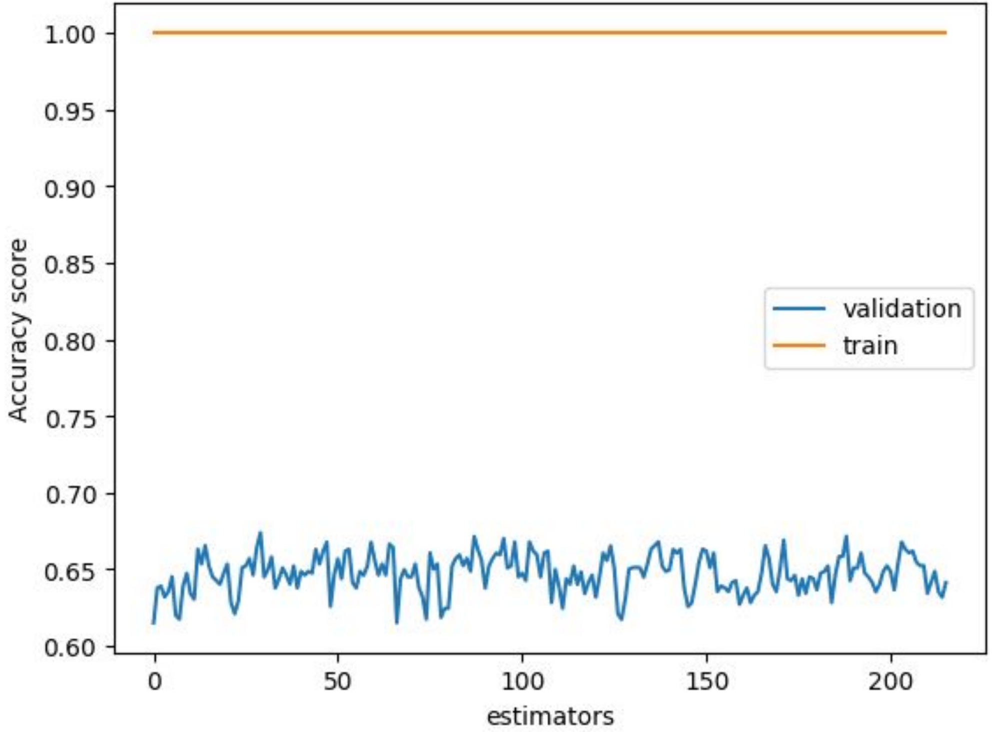
Image source: geeksforgeeks

Algorithms: Gradient Boost

Results on **Non-Engineered Features**

Tuning hyperparameter max_depth and min_samples_leaf, learning_rate, max_features and n_estimators:

- Best validation results for max_depth=10, max_features=sqrt, min_samples_leaf=5, learning_rate=0.1, n_estimators=150



Algorithms: Gradient Boost

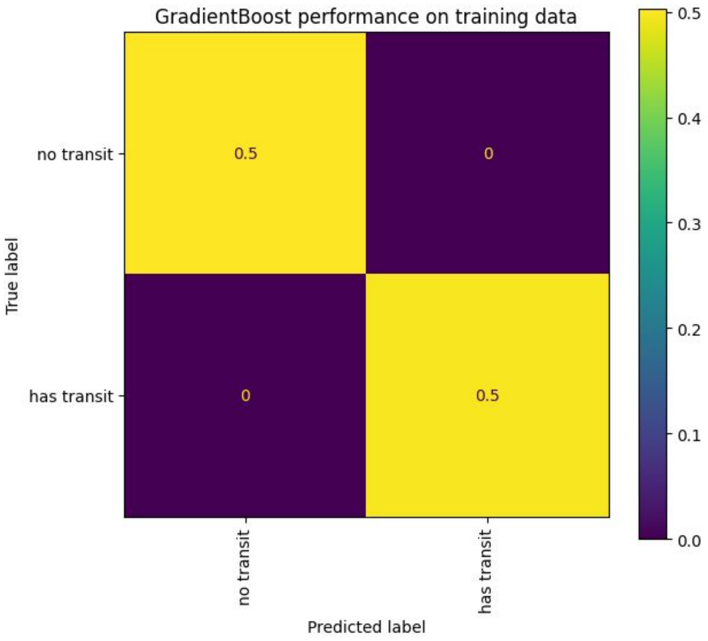
Results on **Non-Engineered Features**

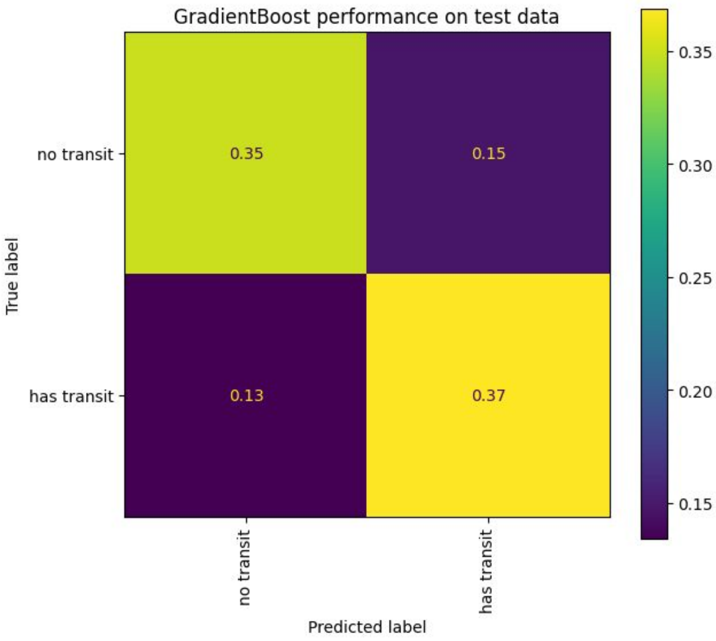
Tuning hyperparameter max_depth and min_samples_leaf, learning_rate, max_features and n_estimators:

- Best validation results for max_depth=10, max_features=sqrt, min_samples_leaf=5, learning_rate=0.1, n_estimators=150

Performance:

```
Validation Data:
        Accuracy 1.0
        Precision 1.0
        Recall 1.0
        F1 1.0
Test Data:
        Accuracy 0.718
        Precision 0.713
        Recall 0.733
        F1 0.723
```



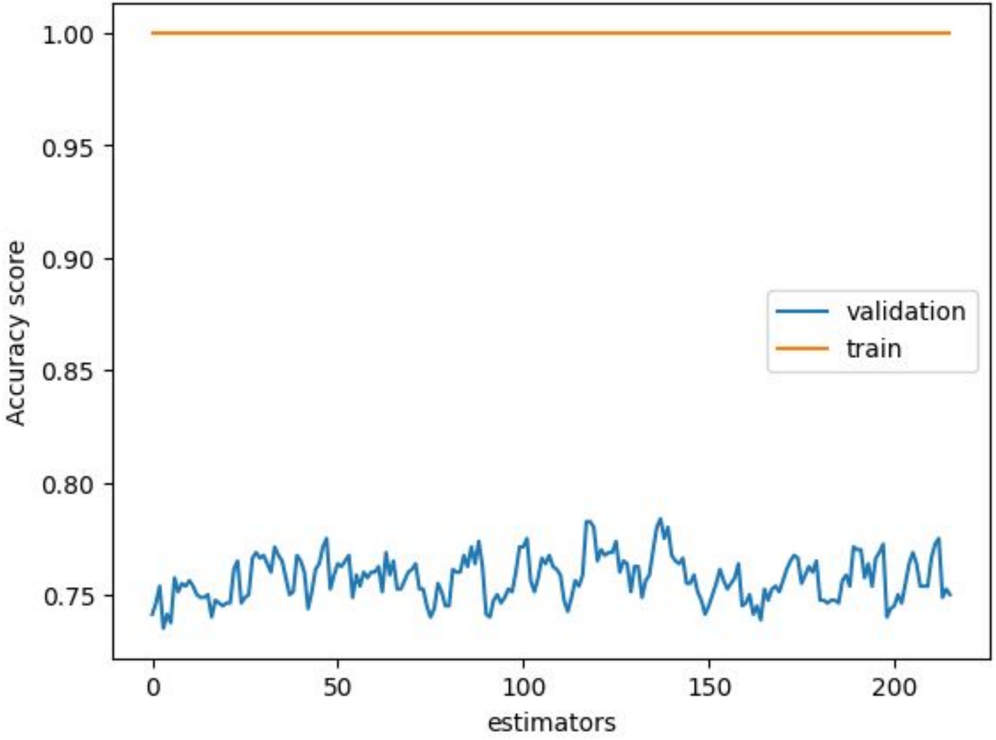


Algorithms: Gradient Boost

Results on **Engineered Features**

Tuning hyperparameter max_depth and min_samples_leaf, learning_rate, max_features and n_estimators:

- Best validation results for max_depth=30, min_samples_leaf=5, max_features=sqrt learning_rate=0.15, n_estimators=150



Algorithms: Gradient Boost

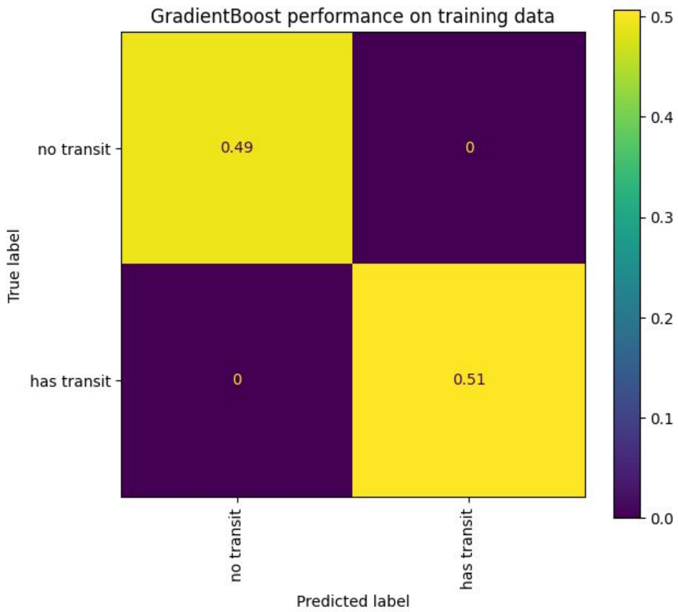
Results on **Engineered Features**

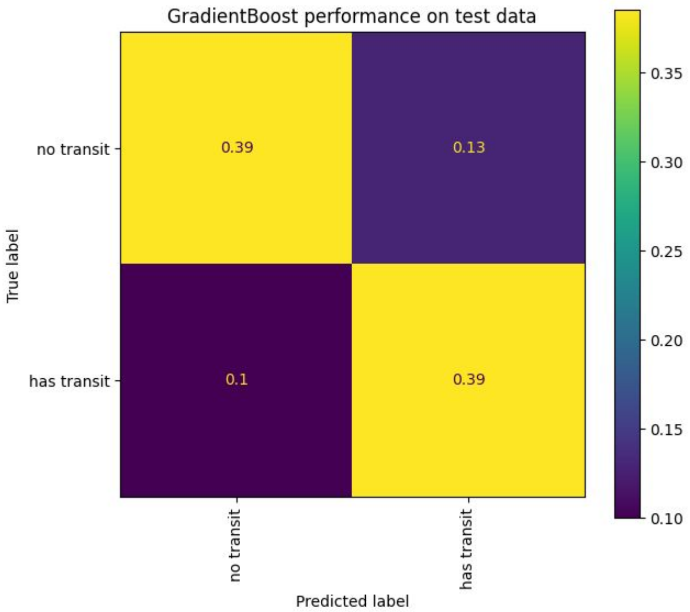
Tuning hyperparameter max_depth and min_samples_leaf, learning_rate, max_features and n_estimators:

- Best validation results for max_depth=30, min_samples_leaf=5, max_features=sqrt learning_rate=0.15, n_estimators=150

Performance:

```
Validation Data:
         Accuracy 1.0
         Precision 1.0
         Recall 1.0
         F1 1.0
Test Data:
         Accuracy 0.77
         Precision 0.748
         Recall 0.794
         F1 0.77
```





Deep Learning

Neural Network = web of nodes (“neurons”) that learn how to weight various learned features of an input dataset to produce the correct output.

Deep Learning

Neural Network = web of nodes (“neurons”) that learn how to weight various learned features of an input dataset to produce the correct output.

Arranged into layers

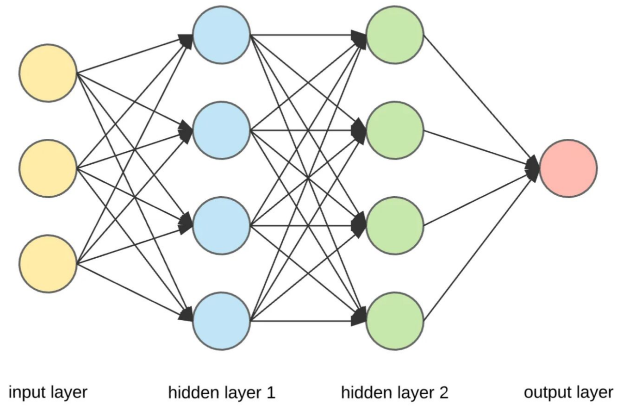


Image from Ognjanovski, 2019

Deep Learning

Neural Network = web of nodes ("neurons") that learn how to weight various learned features of an input dataset to produce the correct output.

Arranged into layers

- Number of nodes of each layer represent number of "features" that layer contains.

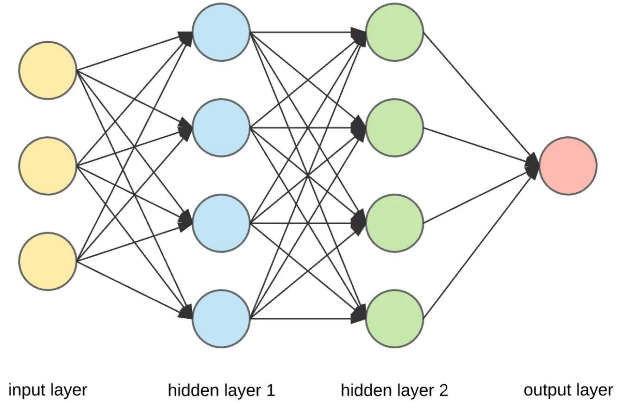


Image from Ognjanovski, 2019

Deep Learning

Neural Network = web of nodes ("neurons") that learn how to weight various learned features of an input dataset to produce the correct output.

Arranged into layers

- Number of nodes of each layer represent number of "features" that layer contains.
- Nodes in given layer connected to some/all in previous one through weights.

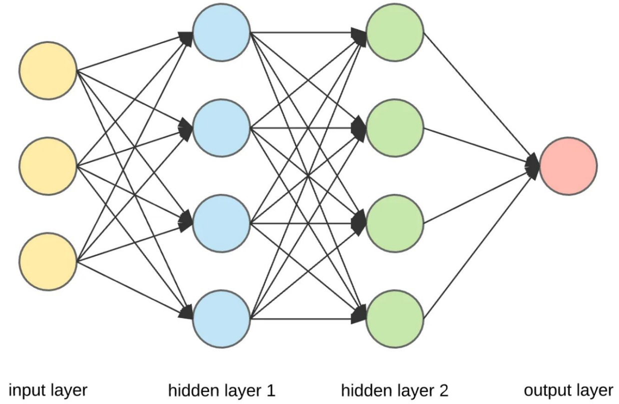


Image from Ognjanovski, 2019

Deep Learning

Neural Network = web of nodes (“neurons”) that learn how to weight various learned features of an input dataset to produce the correct output.

Arranged into layers

- Number of nodes of each layer represent number of “features” that layer contains.
- Nodes in given layer connected to some/all in previous one through weights.
- Layers can also contain "bias" nodes that are not connected to previous layers.

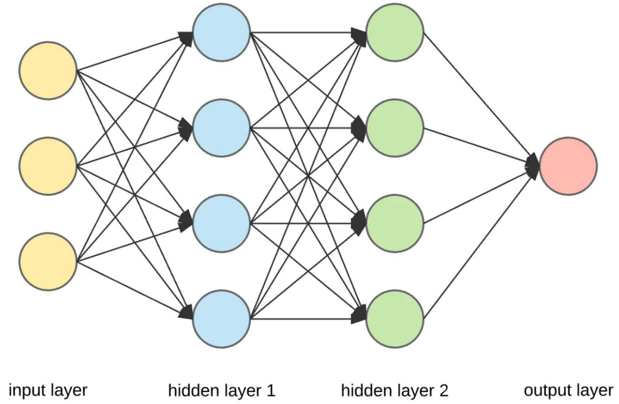


Image from Ognjanovski, 2019

Deep Learning

Neural Network = web of nodes (“neurons”) that learn how to weight various learned features of an input dataset to produce the correct output.

Arranged into layers

- Number of nodes of each layer represent number of “features” that layer contains.
- Nodes in given layer connected to some/all in previous one through weights.
- Layers can also contain "bias" nodes that are not connected to previous layers.

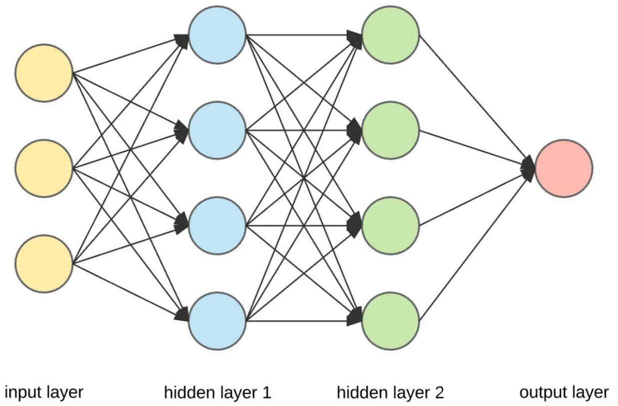


Image from Ognjanovski, 2019

Different networks types use different layer types - “pass” data through the layers differently

Deep Learning

Neural Network = web of nodes ("neurons") that learn how to weight various learned features of an input dataset to produce the correct output.

Arranged into layers

- Number of nodes of each layer represent number of "features" that layer contains.
- Nodes in given layer connected to some/all in previous one through weights.
- Layers can also contain "bias" nodes that are not connected to previous layers.

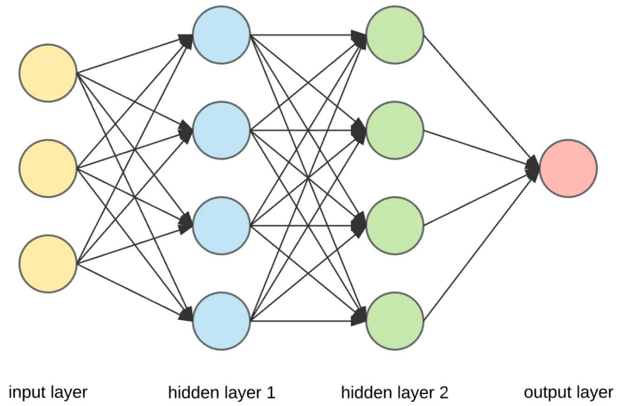


Image from Ognjanovski, 2019

Different networks types use different layer types - "pass" data through the layers differently

End with set of predicted class probabilities; label observation with most probable class.

Deep Learning - training a network

Cost function, C, quantifies difference between predictions and true labels.

Deep Learning - training a network

Cost function, C, quantifies difference between predictions and true labels.

Train using backpropagation

Deep Learning - training a network

Cost function, C, quantifies difference between predictions and true labels.

Train using backpropagation

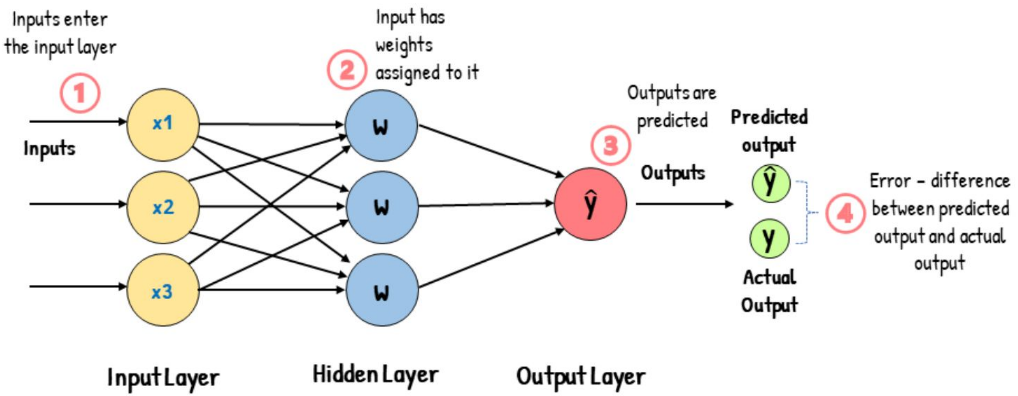


Image source: [Analytics Vidhya]

Deep Learning - training a network

Cost function, C, quantifies difference between predictions and true labels.

Train using backpropagation

- Adjust weights, biases to minimize C.
- Process called “gradient descent”

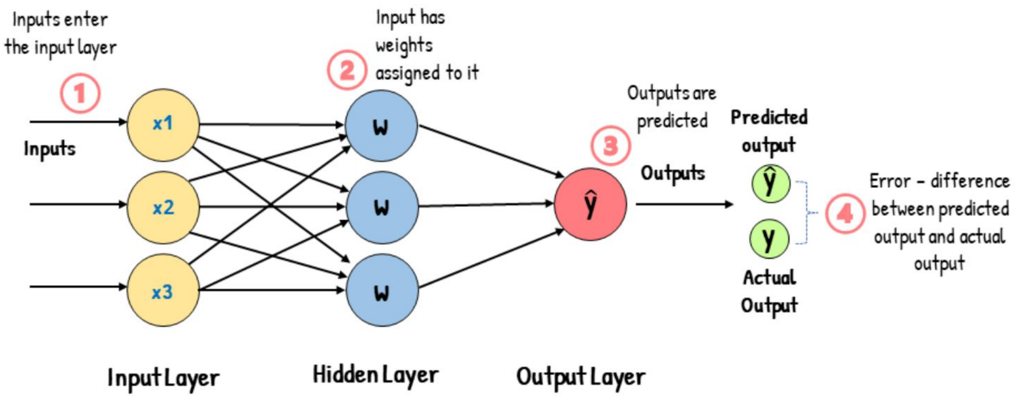


Image source: [Analytics Vidhya]

Deep Learning - training a network

Cost function, C, quantifies difference between predictions and true labels.

Train using backpropagation

- Adjust weights, biases to minimize C.
- Process called “gradient descent”
    - Use chain rule to express how C depends on weights, biases

$$\frac{dC}{dw_i} = \frac{dC}{d\hat{y}} \times \frac{d\hat{y}}{dw_i}$$

$$\frac{dC}{db_i} = \frac{dC}{d\hat{y}} \times \frac{d\hat{y}}{db_i}$$

Deep Learning - training a network

Cost function, C, quantifies difference between predictions and true labels.

Train using backpropagation

- Adjust weights, biases to minimize C.
- Process called “gradient descent”
    - Use chain rule to express how C depends on weights, biases

$$\frac{dC}{dw_i} = \frac{dC}{d\hat{y}} \times \frac{d\hat{y}}{dw_i}$$

$$\frac{dC}{db_i} = \frac{dC}{d\hat{y}} \times \frac{d\hat{y}}{db_i}$$

    - Update each terms using derivatives

$$w_i = w_i - (\alpha \times \frac{dC}{dw_i})$$

$$b_i = b_i - (\alpha \times \frac{dC}{db_i})$$

Learning rate. α, adjusts how fast model learns

Deep Learning - architectures used

First network type: **linear**

Layers connected via simple matrix multiplications.

- Multiply by matrix of weights, θ.
- Weights in θ_i represents connections between nodes in layer i−1 and in layer i.
- Add a bias term, b and apply an activation function, g.

$$\vec{x}_{i+1} = g(\theta \cdot \vec{x}_i + b)$$

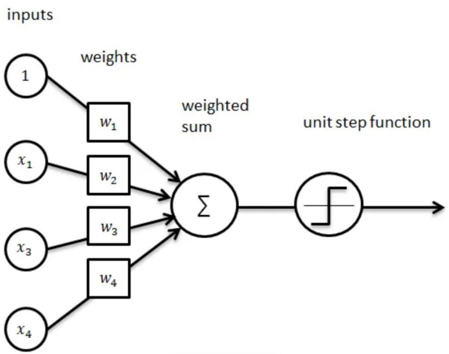


Image from Ognjanovski, 2019

Deep Learning - architectures used

Second network type: **convolutional**

Layers connected via convolutions

- Convolve with kernel (matrix) of given size
- Computed via cross-correlation
- Add a bias term, b and apply an activation function, g.

$$\vec{x}_{i+1} = g(b + \sum_{k}^{n_{nodes-1}} w_{k,i} * x_{k,i-1})$$

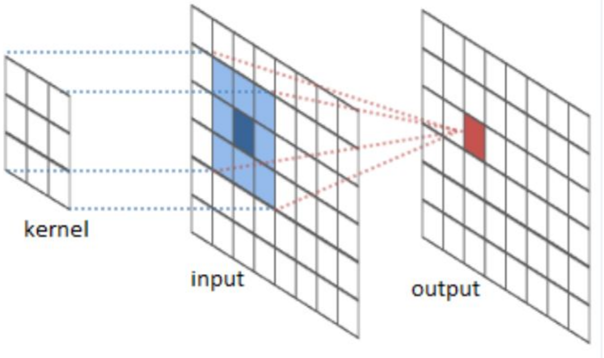


Image from Ognjanovski, 2019

Deep Learning - architectures used

Linear networks

1. Trained on full lightcurves
2. Trained on phase-folded lightcurves

Single-channel convolutional network

3. Trained on phase-folded lightcurves

Double-channel convolutional networks

4. Trained on phase-folded lightcurves and phase-steps

LSTM network

5. Trained on phase-folded lightcurves

Deep Learning - architectures used

Linear networks

1. Trained on full lightcurves

   Poor results - overall accuracy around 45% and recall 29%. F1 of 0.19.
   Why? Perhaps there is too much noise in the curves themselves.
   Thus, we train a second linear model on phase-folded lightcurves.

Deep Learning - architectures used

Linear networks

1. Trained on full lightcurves (~45% acc, ~29% recall, F1~0.19)

2. Trained on phase-folded lightcurves

   Still low very low accuracy (~59%) and recall (~36%). F1 of 0.23.
   Why? Perhaps we need a model that can use the “spatial” (sequential) information better.

Deep Learning - architectures used

Linear networks

1. Trained on full lightcurves (~45% acc, ~29% recall, F1~0.19)
2. Trained on phase-folded lightcurves (~59% acc, ~36% recall, F1 ~0.23)

Single-channel convolutional network

3. Trained on phase-folded lightcurves

   Similarly poor accuracy (~57%), slight increase in recall ~54%, F1 of 0.28.
   Perhaps overall accuracy can be improved by including time-step information.

Deep Learning - architectures used

Linear networks

1. Trained on full lightcurves (~45% acc, ~29% recall, F1~0.19)
2. Trained on phase-folded lightcurves (~59% acc, ~36% recall, F1 ~0.23)

Single-channel convolutional network

3. Trained on phase-folded lightcurves (~57% acc, ~54% recall, F1~0.28)

Double-channel convolutional networks

4. Trained on phase-folded lightcurves and phase-steps

   Significantly better (still poor, ~62%) accuracy, excellent recall (~99%). F1 of ~0.36
   Could this be improved further by other models?

Deep Learning - architectures used

Linear networks

1. Trained on full lightcurves (~45% acc, ~29% recall, F1~0.19)
2. Trained on phase-folded lightcurves (~59% acc, ~36% recall, F1 ~0.23)

Single-channel convolutional network

3. Trained on phase-folded lightcurves (~57% acc, ~54% recall, F1~0.28)

Double-channel convolutional networks

4. Trained on phase-folded curves and phase-steps (~62% acc, ~99% recall, F1~0.32)

LSTM (type of RNN) network

5. Trained on phase-folded lightcurves

   Total accuracy way down to ~49%, but recall ~100%. F1 ~0.33.

Deep Learning - architectures used

Linear networks

1. Trained on full lightcurves (~45% acc, ~29% recall, F1~0.19)
2. Trained on phase-folded lightcurves (~59% acc, ~36% recall, F1 ~0.23)

Single-channel convolutional network

3. Trained on phase-folded lightcurves (~57% acc, ~54% recall, F1~0.28)

Double-channel convolutional networks

4. Trained on phase-folded curves and phase-steps (~62% acc, ~99% recall, F1~0.32)

LSTM (type of RNN) network

5. Trained on phase-folded lightcurves (~49% acc, ~100% recall, F1~0.33)

Deep Learning - overall thoughts

- LSTM has the best F1 and recall
  - But super low accuracy likely means we are just predicting way to many transits
- 2-channel CNN has much better accuracy and still extremely high recall. Similar F1.
  - Appears to be the best NN model so far
- All NN models do worse than simpler algorithms, so we turn back to those to select our best model.

Feature selection

Feature selection in refers to the process of selecting a subset of relevant and significant features (input variables) from the original set of features in a dataset. Of various methods of feature selection we tired out Recursive feature selection(RFE) and Filter Method particularly sklearn selectKBest.

Recursive feature selection(RFE): It is an iterative feature selection method that starts with all features and repeatedly removes the least important features based on a model's performance until a desired number of features is reached.

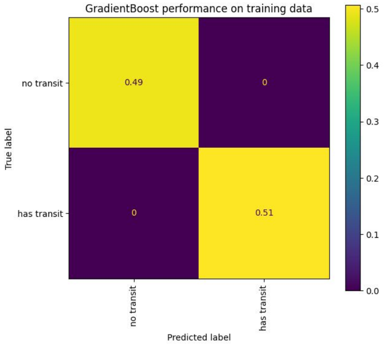
Since the number of features in our dataset was very large this method took too long to complete.

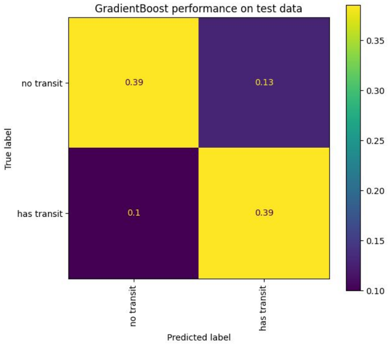
Filter Method: Filter methods pick up the intrinsic properties of the features measured via univariate statistics instead of cross-validation performance. They evaluate the relevance of features based on their individual characteristics, such as correlation with the target variable or statistical tests like ANOVA or chi-square tests.

We experimented with the SelectKBest API to reduce the number of features. We used the default scrore_function of f_classif. After trying different number of features the overall performance of the models did not improve.

Best Model Performance: Gradient Boost Classifier

```
Validation Data:
        Accuracy 1.0
        Precision 1.0
        Recall 1.0
        F1 1.0
Test Data:
        Accuracy 0.77
        Precision 0.748
        Recall 0.794
        F1 0.77
```





Can we do better than this?
It is likely that our limitation is the data itself.

Future Work

Classification of lightcurves is challenging

- Difficult to extract the **low signal-to-noise** transits of small planets
- **Stellar variability** can mimic periodic transit signals
- Very **high dimensional** data

Future work could explore **new classification methods** better designed for time-series data:

- AR(I)MA - Autoregressive (Integrated) Moving Average
- Deep Learning
  - Convolutional Neural Networks
  - Feedforward Neural Networks

Or, explore **new methods of feature engineering:**

- Fit transit models and extract transit parameters as features
- Transform lightcurves using Dynamic Time Warping or Wavelet Analysis

Group Contributions

Initial **data acquisition/exploration**

- Select dataset, create function for downloading and visualizing example observations → **Anna**
- Create function to download and standardize all observations → **Leah**
- Literature review of past works → **Leah**

Preprocessing

- Stitching/detrending/smoothing observations, filling in missing values → **Anna**

Feature engineering

- Phase-folding → **Anna**
- Exploration of Recursive Feature Selection and SelectKBest → **Ashutosh**

Implementation/analysis of **ML algorithms**

- Standard algorithms: KNN, RF, and LR → **Anna**
- Standard algorithms: SVM, AdaBoost, and Gradient Boost → **Ashutosh**
- Deep Learning: Linear Neural Network → **Leah**
- Deep Learning: Convolutional Neural Network → **Leah**

Project **administration**

- Proposal write-up → **Anna**, **Leah, Ashutosh**
- Checkpoint write-up → **Anna**, **Leah**
- Final presentation write-up → **Anna**, **Leah**, **Ashutosh**
- Final slides → **Anna**, **Leah**, **Ashutosh**, **Andrew**

Works Cited

S. Aigrain and F. Favata. Bayesian detection of planetary transits. A modified version of the Gregory-Loredo method for Bayesian periodic signal detection. , 395:625–636, Nov. 2002. doi: 10.1051/0004-6361:20021290.

M. Ansdell, et al. Scientific domain knowledge improves exoplanet transit classification with deep learning. The Astrophysical Journal, 869(1):L7, dec 2018. doi: 10.3847/2041-8213/aaf23b.

D. J. Armstrong, et al. machine learning classification of variable stars and eclipsing binaries in k2 fields 0–4. Monthly Notices of the Royal

Astronomical Society , 456(2):2260–2272, dec 2015. doi: 10.1093/mnras/stv2836.

L. F. A. Arnold. Transit Light-Curve Signatures of Artificial Objects. , 627(1):534–539, July 2005. Doi: 10.1086/430437.

N. M. Batalha, et al. SELECTION, PRIORITIZATION, AND CHARACTERISTICS OF iKEPLER/i TARGET STARS. The Astrophysical Journal , 713(2):L109–L114, mar 2010. doi: 10.1088/2041-8205/713/2/l109. J. Catanzarite. Autovetter Planet Candidate Catalog for Q1-Q17 Data Release 24.

Astronomy and Astro-physics, July 2015.

D. Charbonneau, T. M. Brown, D. W. Latham, and M. Mayor. Detection of Planetary Transits Across a Sun-like Star. , 529(1):L45–L48, Jan. 2000. doi: 10.1086/312457

P. Chintarungruangchai and I.-G. Jiang. Detecting exoplanet transits through machine-learning techniques with convolutional neural networks. Publications of the Astronomical Society of the Pacific, 131(1000): 064502, may 2019. doi: 10.1088/1538-3873/ab13d3.

J. L. Coughlin, et al.. Planetary Candidates Observed by Kepler. VII. The First Fully Uniform Catalog Based on the Entire 48-month Data Set (Q1-Q17 DR24). , 224(1):12, May 2016. doi: 10.3847/0067-0049/224/1/12.

S. Grziwa, M. P¨atzold, and L. Carone. The needle in the haystack: searching for transiting extrasolar planets in CoRoT stellar light curves. , 420(2):1045–1052, Feb. 2012. doi: 10.1111/j.1365-2966.2011.19970.x.

J. M. Jenkins, et. al.Overview of the Kepler Science Processing Pipeline. , 713(2):L87–L91, Apr. 2010. doi: 10.1088/2041-8205/713/2/L87.

D. M. Kipping and A. Teachey. A cloaking device for transiting planets. , 459:1233–1241, June 2016. Doi: 10.1093/mnras/stw672.

G. Kov´acs, S. Zucker, and T. Mazeh. A box-fitting algorithm in the search for periodic transits. , 391: 369–377, Aug. 2002. doi: 10.1051/0004-6361:20020802.

D. M. Kipping and A. Teachey. A cloaking device for transiting planets. , 459:1233–1241, June 2016. Doi: 10.1093/mnras/stw672.

G. Kov´acs, S. Zucker, and T. Mazeh. A box-fitting algorithm in the search for periodic transits. , 391: 369–377, Aug. 2002. doi: 10.1051/0004-6361:20020802.

A. Malik, B. P. Moster, and C. Obermeier. Exoplanet detection using machine learning. Monthly Notices of the Royal Astronomical Society, dec 2021. doi: 10.1093/mnras/stab3692.

S. D. McCauliff, et. al. AUTOMATIC CLASSIFICATION OFiKEPLER/iPLANETARY TRANSIT CANDIDATES. The Astrophysical Journal, 806(1):6, jun 2015. doi: 10.1088/0004-637x/806/1/6.

Ognjanovski, G. (2019, January 14). Everything you need to know about Neural Networks and Backpropagation — Machine Learning Easy and Fun. Towards Data Science. Retrieved July 24, 2023, from https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a

K. A. Pearson, L. Palafox, and C. A. Griffith. Searching for exoplanets using artificial intelligence. , 474(1): 478–491, Feb. 2018. doi: 10.1093/mnras/stx2761.

G. R. Ricker, et al. Transiting Exoplanet Survey Satellite (TESS). Journal of Astronomical Telescopes, Instruments, and Systems, 1:014003, Jan. 2015. Doi: 10.1117/1.JATIS.1.1.014003.

N. Schanche, et al. Machine-learning approaches to exoplanet transit detection and candidate validation in wide-field ground-based surveys. Monthly Notices of the Royal Astronomical Society, 483(4): 5534–5547, nov 2018. doi: 10.1093/mnras/sty3146.

C. J. Shallue and A. Vanderburg. Identifying exoplanets with deep learning: A five-planet resonant chain around kepler-80 and an eighth planet around kepler-90. The Astronomical Journal, 155(2):94, jan 2018. doi: 10.3847/1538-3881/aa9e09.

M. C. Stumpe, J. C. Smith, J. H. Catanzarite, J. E. Van Cleve, J. M. Jenkins, J. D. Twicken, and F. R.

Girouard. Multiscale Systematic Error Correction via Wavelet-Based Bandsplitting in Kepler Data. , 126 (935):100, Jan. 2014. doi: 10.1086/674989.

S. Thompson, D. Fraquelli, J. E. van Cleve, and D. A. Caldwell. Kepler: A search for terrestrial planets (kepler archive manual). may 2016.

A. Wolszczan and D. A. Frail. A planetary system around the millisecond pulsar PSR1257 + 12. , 355 (6356):145–147, Jan. 1992. doi: 10.1038/355145a0.

L. Yu, et al. Identifying exoplanets with deep learning. III. automated triage and vetting of iTESS/i candidates. The Astronomical Journal , 158(1):25, jun 2019. doi: 10.3847/1538-3881/ab21d6.

S. Zucker and R. Giryes. Shallow Transits—Deep Learning. I. Feasibility Study of Deep Learning to Detect Periodic Transits of Exoplanets. , 155(4):147, Apr. 2018. doi: 10.3847/1538-3881/aaae05.