Nearest Neighbors

Nearest Neighbors

Each data point we have has
been represented using features

Stock, Date, [CalendarFeatures,
Momentum, Jump, Features of
Related stocks], Label

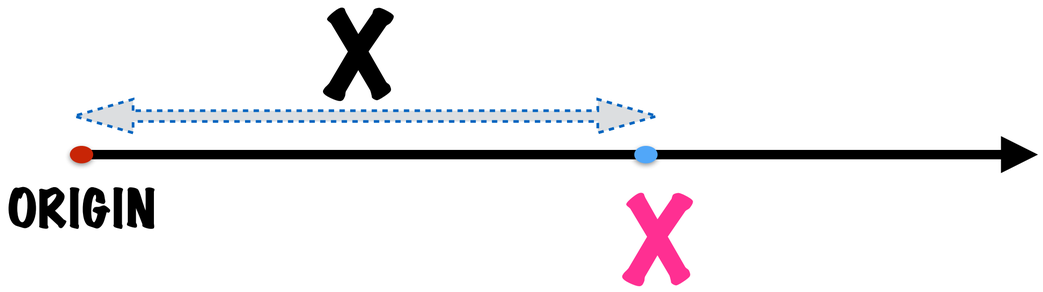
Nearest Neighbors

We can imagine that these features together, represent a point in an N-dimensional hypercube

Stock, Date, [CalendarFeatures, Momentum, Jump, Features of Related stocks], Label

N-Dimensional Hypercube

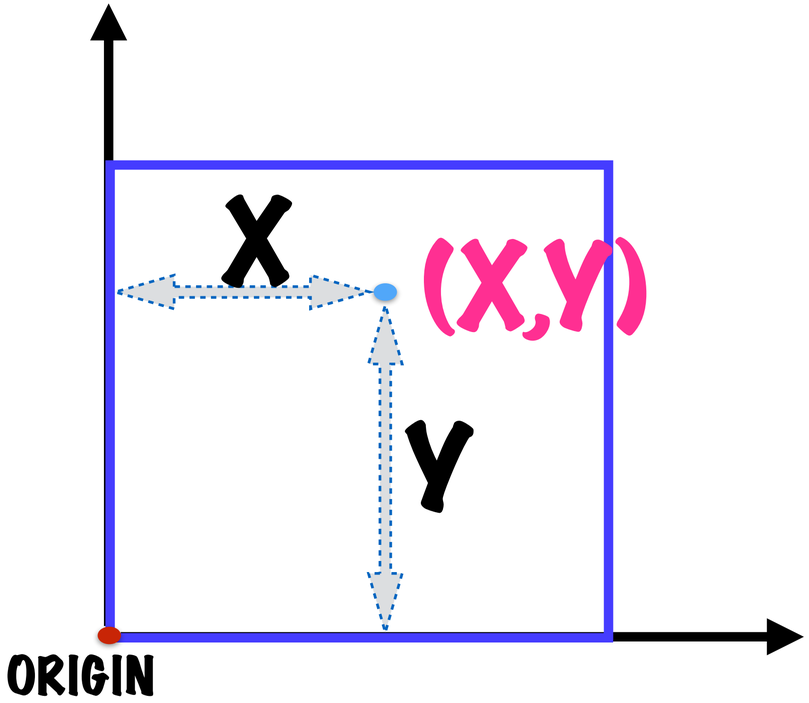
A LINE IS A 1 DIMENSIONAL SHAPE



Any point on a line can be represented using 1 number

N-Dimensional Hypercube

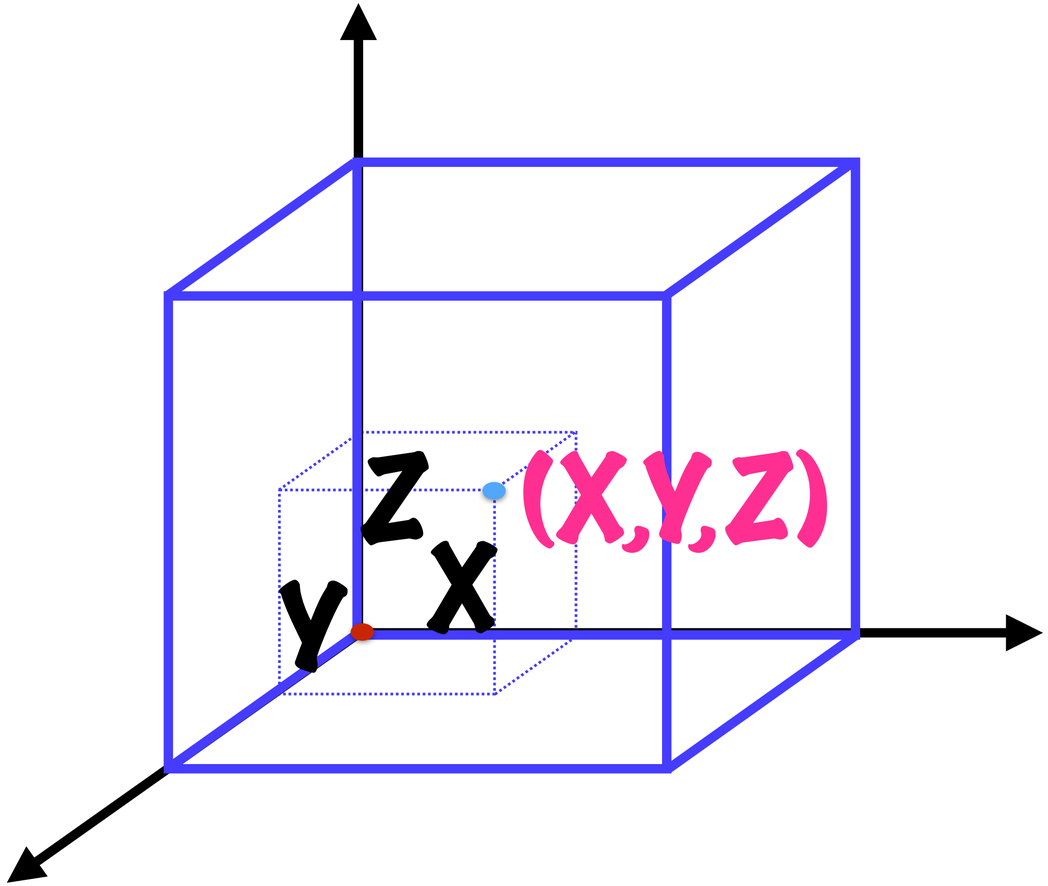
A SQUARE IS A 2 DIMENSIONAL SHAPE



Any point in a square can be represented using 2 numbers

N-Dimensional Hypercube

A CUBE IS A 3 DIMENSIONAL SHAPE



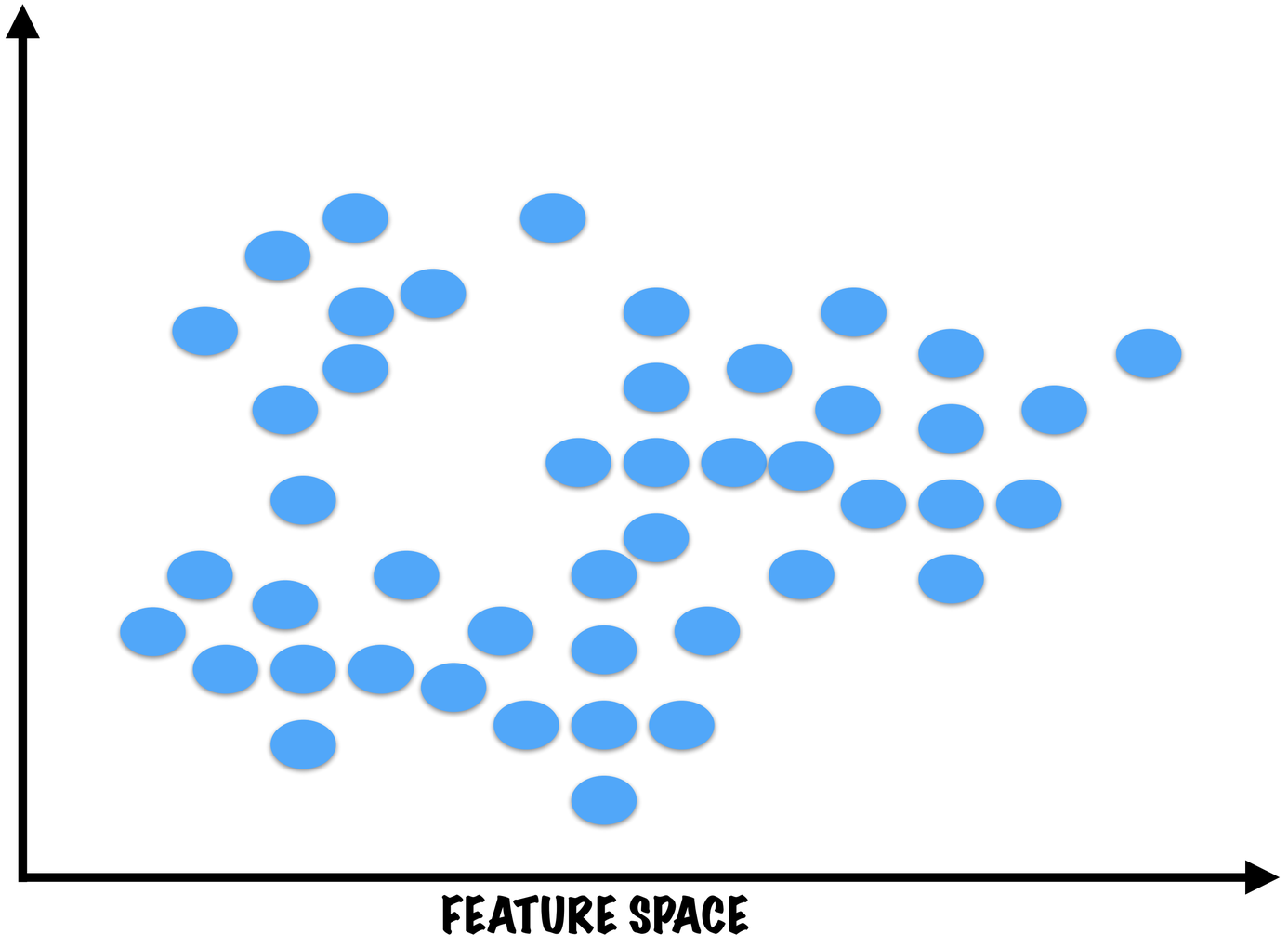
Any point in a cube can be represented with 3 numbers

Nearest Neighbors

A set of N numbers represents a point in an N-Dimensional Hypercube

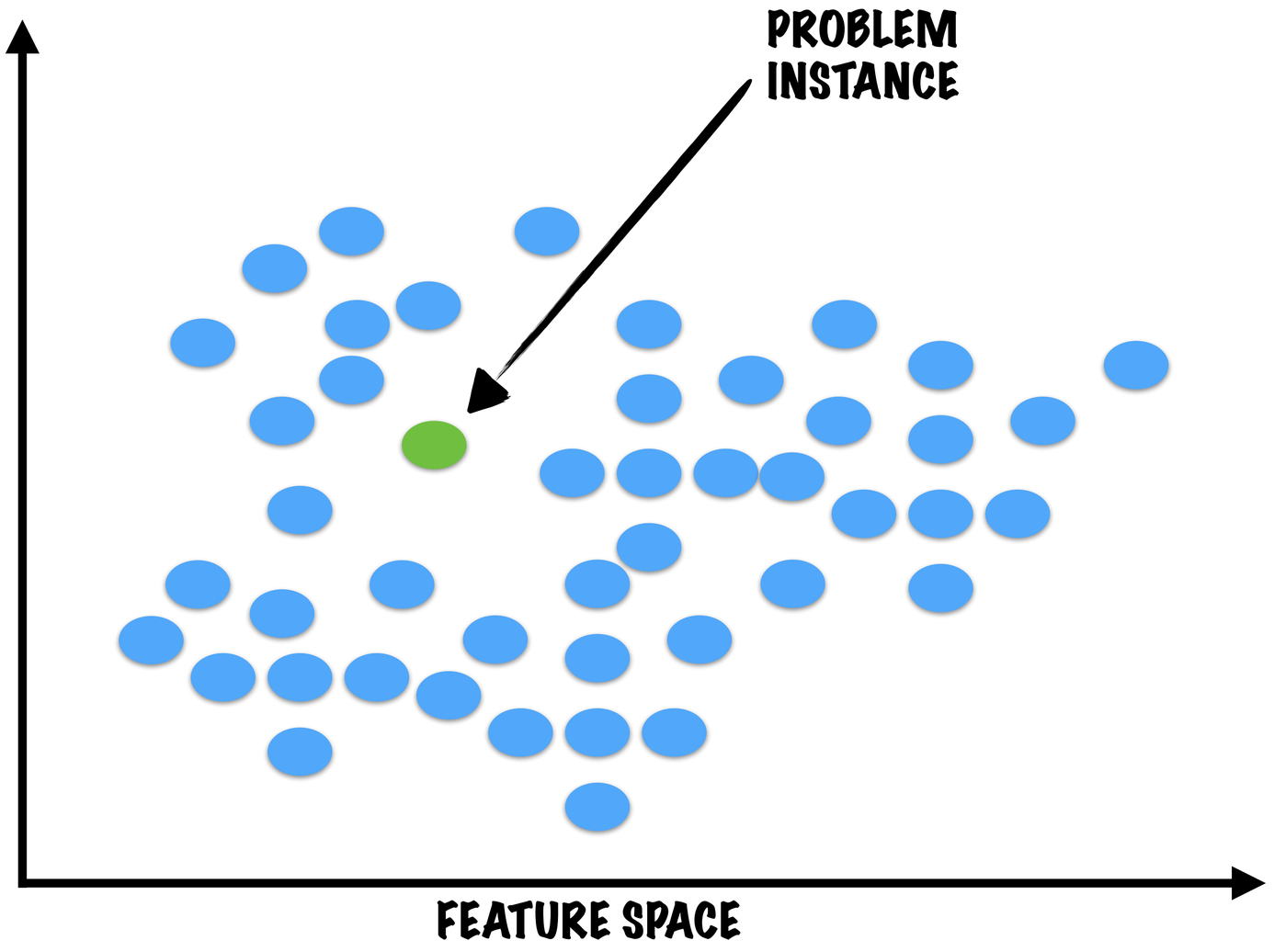
This is just to say that any data in the world can be represented as points in an N-Dimensional space

Nearest Neighbors



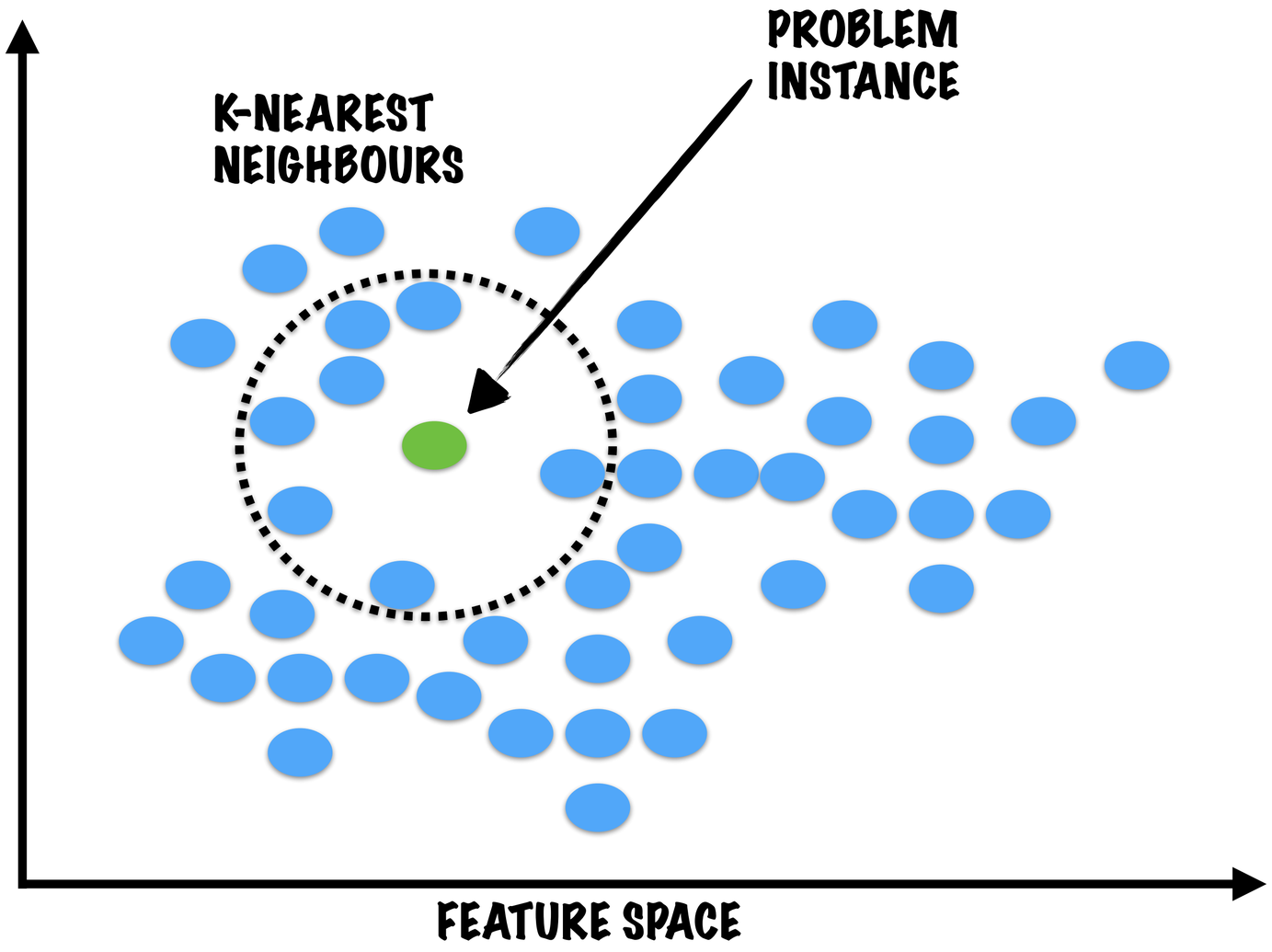
All the past data can be represented as points in this N-Dimensional Space

Nearest Neighbors



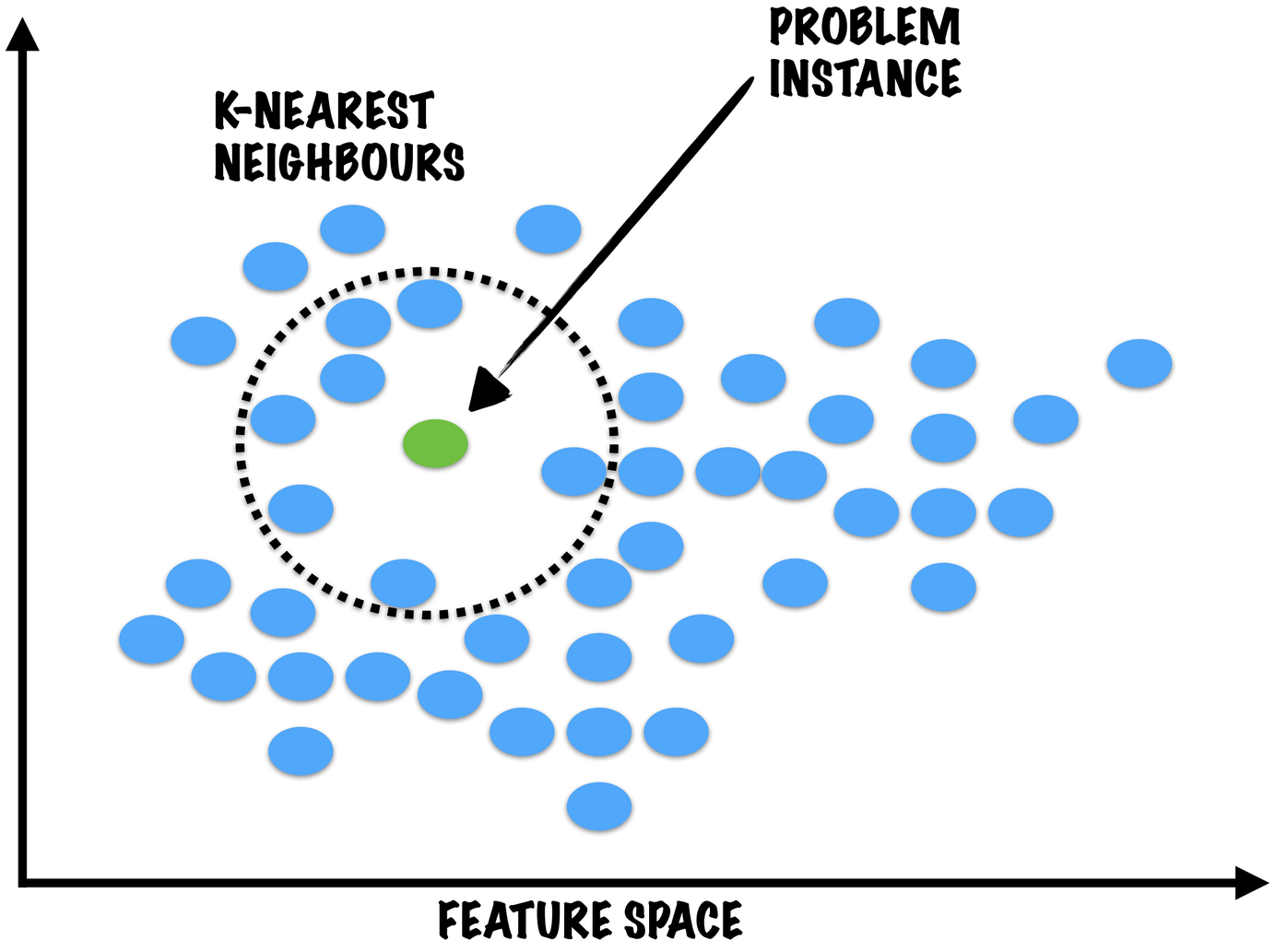
When we have a new date, we can also represent that point in this space

Nearest Neighbors



The returns on the new date can be estimated using the nearest neighbors

Nearest Neighbors



Predicted Return =

The weighted average return of the 5 nearest neighbours

Nearest Neighbors

The trading strategy - i.e. long or short can be decided based on the predicted return

Nearest Neighbors

KNeighborsRegressor

This model in Scikit Learn can compute the average of the labels using the specified number of nearest neighbors

Nearest Neighbors

KNeighborsRegressor

This model in Scikit Learn can compute the **average of the labels** using the specified number of nearest neighbors

Here the average of the labels is nothing but the average of returns of nearest dates

Nearest Neighbors

KNeighborsRegressor

This model in Scikit Learn can compute the average of the labels using the specified number of **nearest neighbors**

The nearest neighbors are the dates in the past most similar to our test date

Nearest Neighbors

KNeighborsRegressor

This model in Scikit Learn can compute the average of the labels using the specified number of **nearest neighbors**

The nearer a past date is, the more similar the features of that date are to our test date

Gradient Boosted Trees

Gradient Boosted Trees

Just like Random Forests, Gradient Boosted Trees method uses an ensemble of Decision Trees

Gradient Boosted Trees

There are 2 techniques involved here

Ensemble learning with boosting

Gradient Descent

Gradient Boosted Trees

There are 2 techniques involved here

Ensemble learning with boosting

Gradient Descent

RECAP

ENSEMBLE LEARNING

INVOLVES THE USE OF MULTIPLE LEARNERS
AND COMBINING THEIR RESULTS

RECAP

THE IDEA OF ENSEMBLE LEARNING IS SIMPLE..

MODELS TEND TO OVERFIT

IF YOU TRAIN MULTIPLE MODELS

THE OVERFITTING COMPONENTS OF EACH OF THE MODELS WOULD BE DIFFERENT

WHEN YOU COMBINE THESE MODELS

THE OVERFITTING COMPONENTS OF THE MODELS WOULD CANCEL EACH OTHER OUT

AND YOU ARE LEFT WITH THE COMPONENTS THAT REALLY DESCRIBE YOUR DATA

RECAP

A MACHINE LEARNING ENSEMBLE IS A COLLECTION OF MODELS

THE MODELS IN THE ENSEMBLE CAN BE

BASED ON DIFFERENT TECHNIQUES

A COLLECTION WITH 1 SVM, 1 DECISION TREE, 1 NAIVE BAYES, 1 KNN

TRAINED ON DIFFERENT TRAINING SETS

A COLLECTION OF SVMS, EACH TRAINED ON A DIFFERENT TRAINING SET

USING DIFFERENT FEATURES

A COLLECTION OF DECISION TREES, EACH GIVEN A DIFFERENT SET OF FEATURES

USING DIFFERENT VALUES OF PARAMETERS

A COLLECTION OF K-NEAREST NEIGHBOURS, EACH WITH A DIFFERENT VALUE OF K

RECAP

AN ENSEMBLE LEARNER COMBINES THE RESULTS FROM INDIVIDUAL MODELS

THE FINAL RESULT CAN BE

A MAJORITY VOTE OF THE INDIVIDUAL MODELS

AVERAGE OF THE RESULT FROM INDIVIDUAL MODELS

A WEIGHTED FUNCTION OF THE RESULT FROM INDIVIDUAL MODELS

RECAP

BAGGING

BOOSTING

ARE SPECIAL
ENSEMBLE
LEARNING
TECHNIQUES

BAGGING

(BOOTSTRAP-AGGREGATING)

IS AN ENSEMBLE LEARNING TECHNIQUE THAT WAS DEVELOPED FOR CLASSIFICATION PROBLEMS

EACH MODEL IN THE ENSEMBLE IS TRAINED ON A DIFFERENT TRAINING SET

THESE TRAINING SETS ARE RANDOMLY GENERATED FROM THE ORIGINAL TRAINING SET

FOR THE FINAL RESULT, EACH MODEL IS GIVEN AN EQUAL WEIGHT AND A MAJORITY VOTE IS TAKEN

RECAP

BAGGING

BOOSTING

ARE SPECIAL ENSEMBLE LEARNING TECHNIQUES

Random Forest is an example of a technique that uses bagging

RECAP

BAGGING

BOOSTING

ARE SPECIAL ENSEMBLE LEARNING TECHNIQUES

Gradient Boosted Trees use the concept of Boosting

Gradient Boosted Trees

BOOSTING

IS AN ALGORITHM FOR
ITERATIVELY ADDING LEARNERS
TO THE ENSEMBLE

Gradient Boosted Trees

BOOSTING

is an algorithm for iteratively identifying the trees in the ensemble

Gradient Boosted Trees

BOOSTING

In each iteration a subset of the main training set is chosen for training

Gradient Boosted Trees

BOOSTING

In the first iteration, all the data points have equal probability of being chosen in the training set

Gradient Boosted Trees

BOOSTING

In the each subsequent iteration, the probabilities of being chosen are modified

Gradient Boosted Trees

BOOSTING

Points which have been classified correctly more often have lower weight

Gradient Boosted Trees

BOOSTING

Points which have been misclassified start getting higher weight

Gradient Boosted Trees

BOOSTING

Each subsequent tree that is built, is given a training set that has more of the points that are misclassified

Gradient Boosted Trees

BOOSTING

Finally, we have a collection of trees, each trained on a different training set

Gradient Boosted Trees

BOOSTING

The final classification is a weighted vote of all the trees

Gradient Boosted Trees

BOOSTING

The weight of a tree is proportional to how accurate the tree was on the training set

Gradient Boosted Trees

There are 2 techniques involved here

Ensemble learning with boosting

Gradient Descent

Gradient Boosted Trees

There are 2 techniques involved here

Ensemble learning with boosting

Gradient Descent

Gradient Boosted Trees

Gradient Descent

In addition to Boosting,
Gradient Boosted Trees also use
the concept of **Gradient Descent**

Gradient Boosted Trees

Gradient Descent

In each iteration of boosting

1. A misclassification rate is calculated

2. Weights of training samples are updated based on this rate

Gradient Boosted Trees

Gradient Descent

In each iteration of boosting

This is based on an error function

1. A misclassification rate is calculated

2. Weights of training samples are updated based on this rate

Gradient Boosted Trees

Gradient Descent

The error function is of the form
M(x) + C(x)

In each iteration of boosting

1. A misclassification rate is calculated

2. Weights of training samples are updated based on this rate

M(x) + C(x)

M is the misclassification rate aka the loss

1. A misclassification rate is calculated

2. Weights of training samples are updated based on this rate

M(x) + C(x)

C is the complexity of the trees being constructed

1. A misclassification rate is calculated

2. Weights of training samples are updated based on this rate

M(x) + C(x)

Minimizing this function will minimize the error, at the same time penalizing models with very high complexity

1. A misclassification rate is calculated

2. Weights of training samples are updated based on this rate

M(x) + C(x)

This combination is what allows Boosting to find a good ensemble of trees without overfitting

1. A misclassification rate is calculated

2. Weights of training samples are updated based on this rate

M(x) + C(x)

The weights are updated in such a way that this error function is minimized

2. Weights of training samples are updated based on this rate

M(x) + C(x)

The update is done using Gradient descent , which is an optimization technique

2. Weights of training samples are updated based on this rate

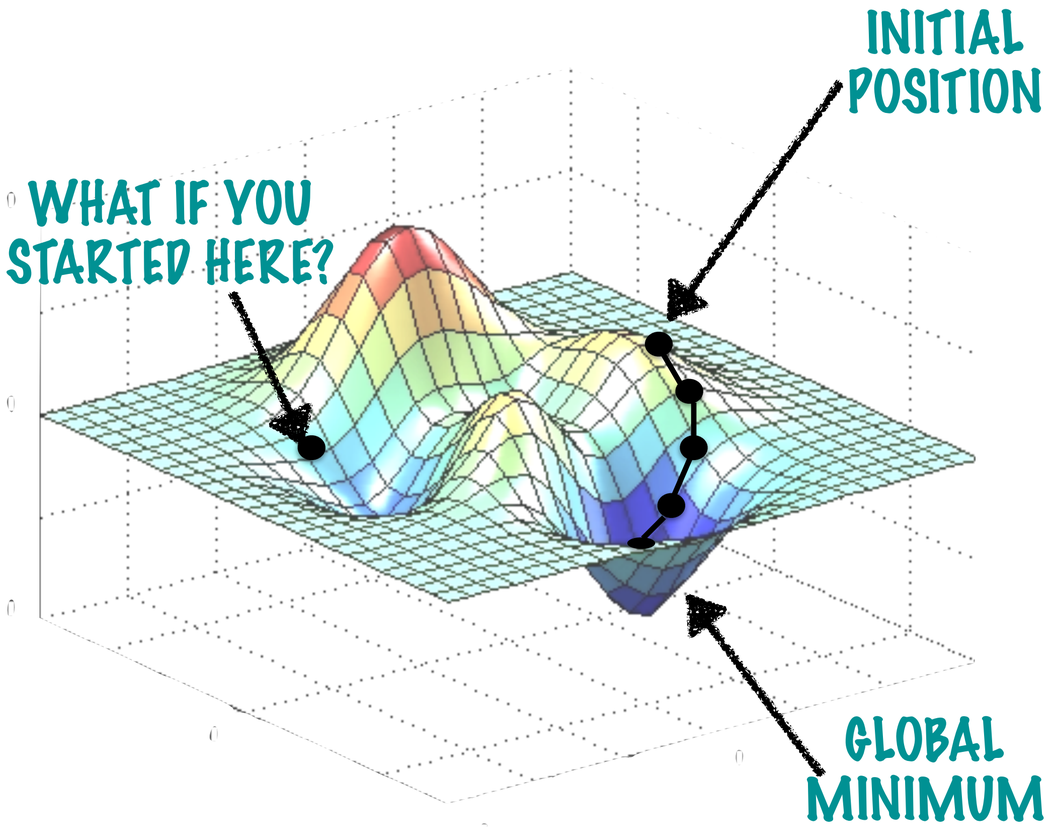
STOCHASTIC GRADIENT DESCENT

M(x) + C(x)

1. FIND THE CURRENT VALUE OF THE ERROR FUNCTION

2. FIND THE SLOPE AT THE CURRENT POINT AND MOVE SLIGHTLY DOWNWARDS IN THAT DIRECTION

3. REPEAT UNTIL YOU REACH A MINIMUM



GRADIENT DESCENT DOESN'T GUARANTEE THE GLOBAL MINIMUM

Gradient Boosted Trees

There are 2 techniques involved here

Ensemble learning with boosting

Gradient Descent

Gradient Boosted Trees

There are 2 techniques involved here

Ensemble learning with boosting

Gradient Descent

Gradient Boosted Trees

Ensemble learning with boosting

Gradient Descent

Gradient Boosting is not limited to only decision trees

Gradient Boosted Trees

Ensemble learning with boosting

Gradient Descent

Using the above 2 techniques a Gradient Boosted Classifier / Regressor can be constructed with any base algorithm

Gradient Boosted Trees

XGBoost

is a library that can be used to build Gradient Boosted Trees

Gradient Boosted Trees

XGBClassifier

is the classifier in this library that is very similar to the Classifiers in ScikitLearn

Gradient Boosted Trees

XGBClassifier

This classifier also has fit and predict methods to train and test a classifier

Gradient Boosted Trees

XGBClassifier

There are several parameters in XGBClassifier that need to be carefully chosen

Gradient Boosted Trees

XGBClassifier

n_estimators

learning_rate

max_depth

min_child_weight

subsample

gamma

colsample_bytree

Gradient Boosted Trees

XGBClassifier

n_estimators

min_child_weight

learning_rate

subsample

max_depth

gamma

colsample_bytree

The number of trees to be built

Gradient Boosted Trees

XGBClassifier

n_estimators

min_child_weight

learning_rate

subsample

max_depth

gamma

colsample_bytree

This should depend on the size of the training set

Gradient Boosted Trees

XGBClassifier

n_estimators

min_child_weight

learning_rate

subsample

max_depth

gamma

colsample_bytree

The larger the training set, the higher the n_estimators can be

Gradient Boosted Trees

XGBClassifier

n_estimators

min_child_weight

learning_rate

subsample

max_depth

gamma

colsample_bytree

After each iteration, weights for the training samples are updated

Gradient Boosted Trees

XGBClassifier

n_estimators

min_child_weight

learning_rate

The learning rate is a factor that the weights are multiplied by

subsample

gamma

max_depth

colsample_bytree

Gradient Boosted Trees

XGBClassifier

n_estimators

min_child_weight

learning_rate

This shrinks the weights on each step

subsample

max_depth

gamma

colsample_bytree

Gradient Boosted Trees

XGBClassifier

n_estimators

min_child_weight

learning_rate

subsample

max_depth

gamma

colsample_bytree

Such a factor can reduce the possibility of overfitting

Gradient Boosted Trees

XGBClassifier

n_estimators

min_child_weight

learning_rate

subsample

max_depth

gamma

colsample_bytree

Typically, the final values to used are in the range of 0.01-0.2

Gradient Boosted Trees

XGBClassifier

The maximum depth of a tree

max_depth

Gradient Boosted Trees

XGBClassifier

This factor is used to control over-fitting

max_depth

Gradient Boosted Trees

XGBClassifier

A higher depth means more complex trees

max_depth

Gradient Boosted Trees

XGBClassifier

The higher the complexity, more the chance to learn relations very specific to a particular sample

max_depth

Gradient Boosted Trees

XGBClassifier

Typically, the value chosen is between 3-10

max_depth

Gradient Boosted Trees

XGBClassifier

min_child_weight

This is another factor that defines the complexity of the tree

n_estimators
learning_rate
max_depth
subsample
gamma
colsample_bytree

Gradient Boosted Trees

XGBClassifier

min_child_weight

In the tree a new child node is added to denote a new condition that has some importance

Gradient Boosted Trees

XGBClassifier

min_child_weight

The child is created by looking at all samples in the training set that satisfy that condition

Gradient Boosted Trees

XGBClassifier

min_child_weight

min_child_weight is Minimum sum of weights of all observations required in a child

Gradient Boosted Trees

XGBClassifier

n_estimators

min_child_weight

learning_rate

subsample

Higher values prevent overfitting

max_depth

gamma

colsample_bytree

Gradient Boosted Trees

XGBClassifier

min_child_weight

However, this parameter requires careful tuning as too high a value can lead to underfitting

Gradient Boosted Trees

XGBClassifier

This denotes the fraction of the training set that should be used to construct each tree

subsample

Gradient Boosted Trees

XGBClassifier

n_estimators min_child_weight learning_rate max_depth gamma colsample_bytree

The lower the value, the more distinct each tree's training set will be

subsample

Gradient Boosted Trees

XGBClassifier

n_estimators

min_child_weight

This is good to avoid overfitting

subsample

gamma

max_depth

colsample_bytree

Gradient Boosted Trees

XGBClassifier

n_estimators min_child_weight learning_rate max_depth gamma colsample_bytree

However if your training set is too small, then a low value can lead to underfitting

subsample

Gradient Boosted Trees

XGBClassifier

n_estimators min_child_weight

Typical values chosen are between 0.5-1

subsample

gamma

colsample_bytree

Gradient Boosted Trees

XGBClassifier

A node in a tree is split
to make new nodes
only if it means the
error function will be
reduced by that split

gamma

Gradient Boosted Trees

XGBClassifier

Gamma specifies the minimum loss reduction required to make a split

gamma

Gradient Boosted Trees

XGBClassifier

Each tree in the ensemble can be built using a random subset of the overall features

colsample_bytree

Gradient Boosted Trees

XGBClassifier

This parameter is the fraction of the features that should be chosen to train each tree

n_estimators
min_child_weight
learning_rate
subsample
max_depth
gamma

colsample_bytree

Gradient Boosted Trees

XGBClassifier

n_estimators

min_child_weight

learning_rate

subsample

max_depth

gamma

Typically, a value between 0.5-1 is chosen

colsample_bytree

Gradient Boosted Trees

XGBClassifier

n_estimators

learning_rate

max_depth

min_child_weight

subsample

gamma

If 0.5 is chosen, then each tree is built using only 50% of the features, and the features are randomly chosen

colsample_bytree

Gradient Boosted Trees

XGBClassifier

The optimum value for each of these parameters has to be chosen by searching over a large number of possible combinations

Gradient Boosted Trees

Hyperopt

Hyperopt is a python library that can do an optimized search over a large number of combinations to minimize an objective

Gradient Boosted Trees

Hyperopt

```
space = {
    'n_estimators': hp.quniform('n_estimators', 100, 1000, 1),
    'learning_rate': hp.quniform('learning_rate', 0.025, 0.5, 0.025),
    'max_depth': hp.quniform('max_depth', 1, 13, 1),
    'min_child_weight': hp.quniform('min_child_weight', 1, 6, 1),
    'subsample': hp.quniform('subsample', 0.5, 1, 0.05),
    'gamma': hp.quniform('gamma', 0.5, 1, 0.05),
    'colsample_bytree': hp.quniform('colsample_bytree', 0.5, 1, 0.05),
    'nthread': 6,
    'silent': 1
}
```

First we define a search space

Gradient Boosted Trees

Hyperopt

```
space = {
    'n_estimators': hp.quniform('n_estimators', 100, 1000, 1),
    'learning_rate': hp.quniform('learning_rate', 0.025, 0.5, 0.025),
    'max_depth': hp.quniform('max_depth', 1, 13, 1),
    'min_child_weight': hp.quniform('min_child_weight', 1, 6, 1),
    'subsample': hp.quniform('subsample', 0.5, 1, 0.05),
    'gamma': hp.quniform('gamma', 0.5, 1, 0.05),
    'colsample_bytree': hp.quniform('colsample_bytree', 0.5, 1, 0.05),
    'nthread': 6,
    'silent': 1
}
```

For each parameter a sequence of possible values is specified

Gradient Boosted Trees

Hyperopt

```
space = {
    'n_estimators': hp.quniform('n_estimators', 100, 1000, 1),
     'learning_rate': hp.quniform('learning_rate', 0.025, 0.5, 0.025),
    'max_depth': hp.quniform('max_depth', 1, 13, 1),
    'min_child_weight': hp.quniform('min_child_weight', 1, 6, 1),
    'subsample': hp.quniform('subsample', 0.5, 1, 0.05),
    'gamma': hp.quniform('gamma', 0.5, 1, 0.05),
    'colsample_bytree': hp.quniform('colsample_bytree', 0.5, 1, 0.05),
    'nthread': 6,
    'silent': 1
}
```

For instance, the number of estimators can be chosen from a sequence [100, 101,...10001

Gradient Boosted Trees

Hyperopt

```
space = {
    'n_estimators': hp.quniform('n_estimators', 100, 1000, 1),
     'learning_rate': hp.quniform('learning_rate', 0.025, 0.5, 0.025),
    'max_depth': hp.quniform('max_depth', 1, 13, 1),
    'min_child_weight': hp.quniform('min_child_weight', 1, 6, 1),
    'subsample': hp.quniform('subsample', 0.5, 1, 0.05),
    'gamma': hp.quniform('gamma', 0.5, 1, 0.05),
    'colsample_bytree': hp.quniform('colsample_bytree', 0.5, 1, 0.05),
    'nthread': 6,
    'silent': 1
}
```

quniform is used to specify that the value of n_estimators is chosen at random from this sequence

Gradient Boosted Trees

Hyperopt

```
space = {
    'n_estimators': hp.quniform('n_estimators', 100, 1000, 1),
     'learning_rate': hp.quniform('learning_rate', 0.025, 0.5, 0.025),
    'max_depth': hp.quniform('max_depth', 1, 13, 1),
    'min_child_weight': hp.quniform('min_child_weight', 1, 6, 1),
    'subsample': hp.quniform('subsample', 0.5, 1, 0.05),
    'gamma': hp.quniform('gamma', 0.5, 1, 0.05),
    'colsample_bytree': hp.quniform('colsample_bytree', 0.5, 1, 0.05),
    'nthread': 6,
    'silent': 1
}
```

All values in the sequence have equal probability of being chosen

Gradient Boosted Trees

Hyperopt

```
fmin(score, space, algo=tpe.suggest,
trials=trials, max_evals=50)
```

fmin() function will then search through possible combinations in this search space until an optimum combination is found

Gradient Boosted Trees

Hyperopt

```
fmin(score, space, algo=tpe.suggest,
trials=trials, max_evals=50)
```

score is user defined
function whose value
needs to be minimized

Gradient Boosted Trees

Hyperopt

```
fmin(score, space, algo=tpe.suggest,
trials=trials, max_evals=50)
```

trials is a variable where the result of each trial will be stored

Gradient Boosted Trees

Hyperopt

```
fmin(score, space, algo=tpe.suggest,
trials=trials, max_evals=50)
```

max_evals is the total number of trials the function will perform before deciding the best combination