

# Tuning ML Models

[ter.ps/389iweek9](https://ter.ps/389iweek9)

# Feature Selection

- We generally drop linearly correlated features - Codelab 2
- Occam's razor principle - for most models, adding additional correlated features adds complexity and does
- Increased chance to overfit
- Selecting proper features is VERY important

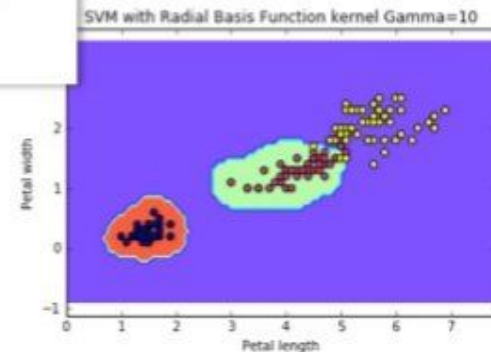
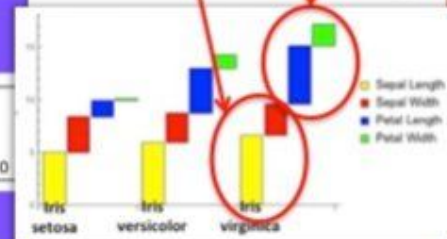
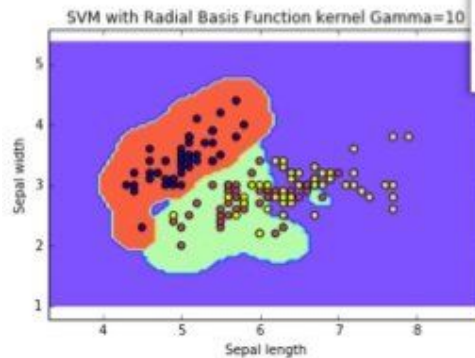
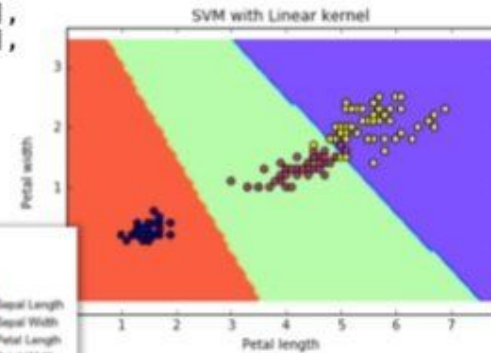
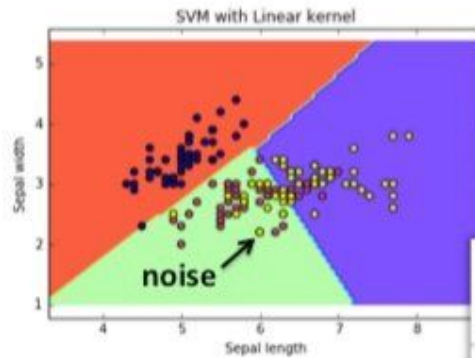


# Variable Selection

Wrong

Right

[ 5.1, 3.5, 1.4, 0.2 ],  
[ 4.9, 3. , 1.4, 0.2 ],  
[ 4.7, 3.2, 1.3, 0.2 ],  
[ 4.6, 3.1, 1.5, 0.2 ],  
[ 5. , 3.6, 1.4, 0.2 ],  
[ 5.4, 3.9, 1.7, 0.4 ],  
[ 4.6, 3.4, 1.4, 0.3 ],



Rubens Zimbres

# Tuning Model Parameters

- Used to fine tune accuracies
- Various types of parameters depending on model
- Look at documentation
- Kernels for SVM, depth & breadth of leaves for Trees, min/max weights for features for Naive Bayes



# Training vs Testing Accuracy

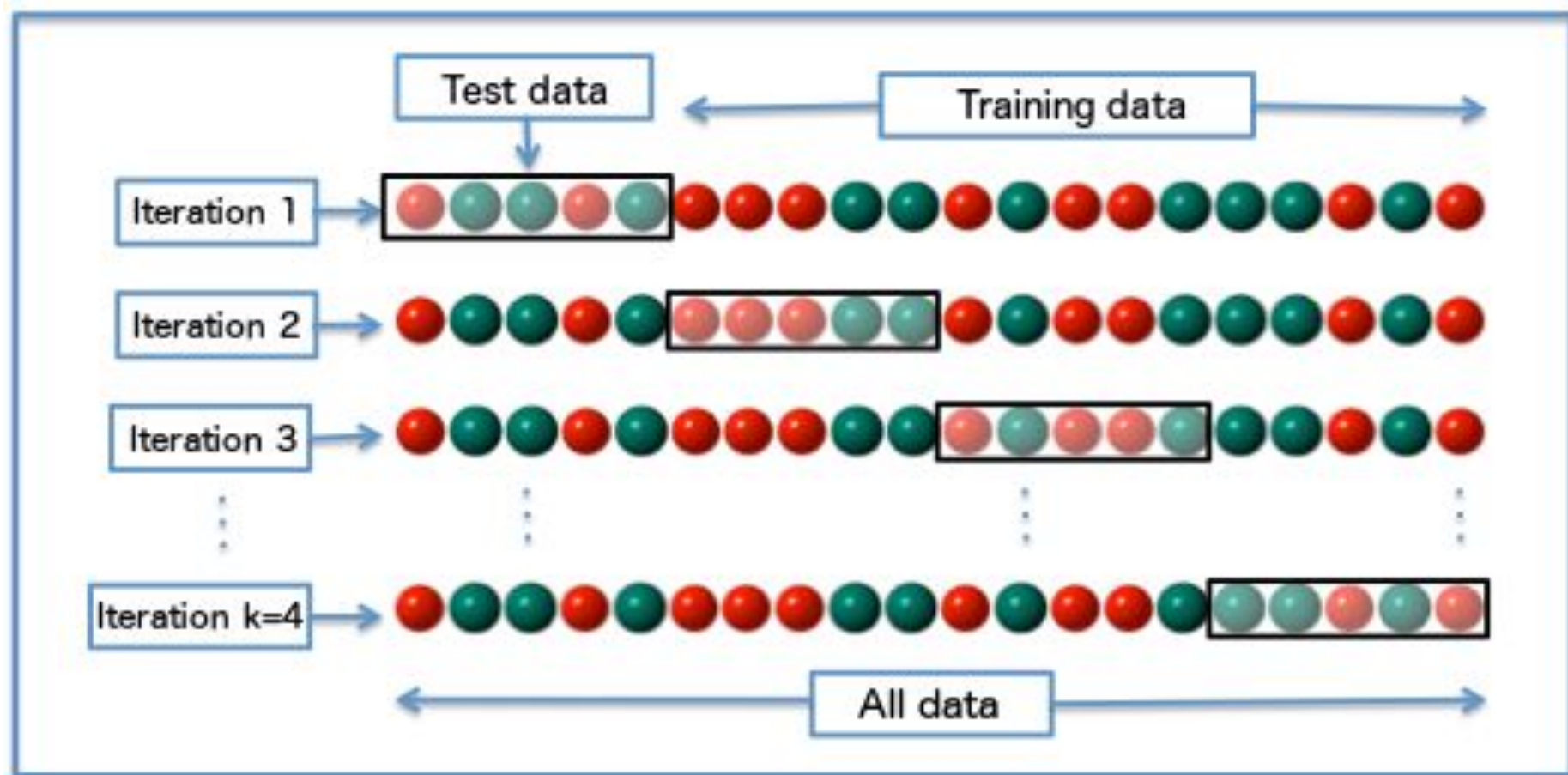
- Want training and testing accuracy as close as possible
- Higher is better, but we don't want a big gap between training and testing data



# Cross-Validation

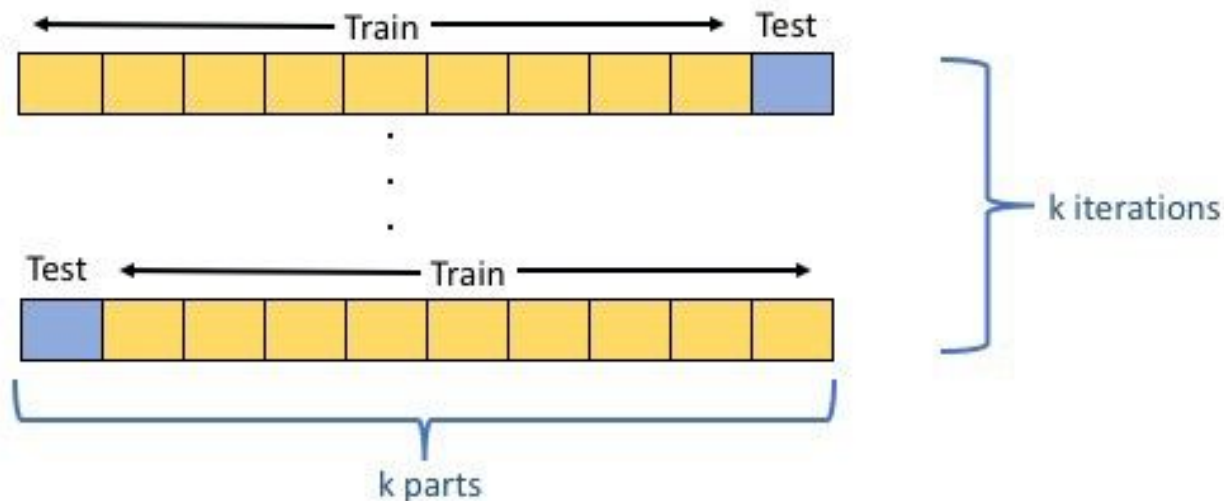
- Codelab 1 examples - K Folds
- Make sure model is fitted properly, generalizes
- Types:
  - K Folds, holdout method, leave one out, leave n out, etc





# K Folds Cross Validation Method

1. Divide the sample data into  $k$  parts.
2. Use  $k-1$  of the parts for training, and 1 for testing.
3. Repeat the procedure  $k$  times, rotating the test set.
4. Determine an expected performance metric (mean square error, misclassification error rate, confidence interval, or other appropriate metric) based on the results across the iterations

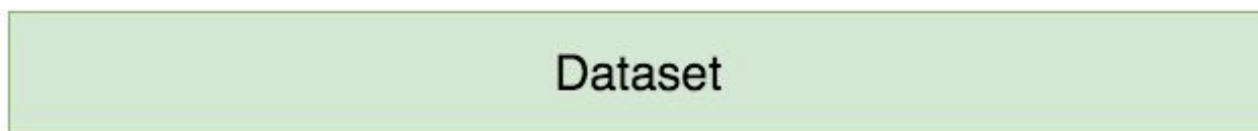




# Training vs. Testing vs. Validation data

- Codelabs so far have only used Training/Testing Sets
- Training Data
  - Model uses this data to “learn” and create predictions
- Testing Data
  - used to test effectiveness of model on new data
- Validation Data
  - Part of the dataset is withheld to use to stress test model
- Another way to ensure you do not underfit/overfit

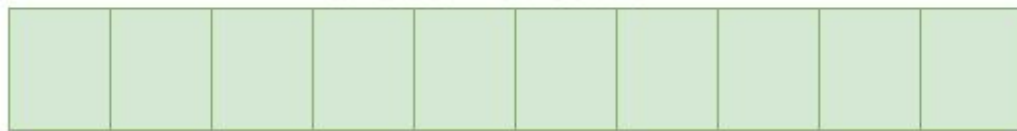




Holdout Method



Cross Validation



Data Permitting:



Training, Validation, Testing



# Tuning

Parameter tuning is the final step in machine learning

Search optimization problem

Algorithm parameters, a.k.a. *hyperparameters*

Grid search and random search



# Grid Search

Methodically build & evaluate model for each combination of algorithm parameters specified in a grid



```
1 # Grid Search for Algorithm Tuning
2 import numpy as np
3 from sklearn import datasets
4 from sklearn.linear_model import Ridge
5 from sklearn.model_selection import GridSearchCV
6 # load the diabetes datasets
7 dataset = datasets.load_diabetes()
8 # prepare a range of alpha values to test
9 alphas = np.array([1,0.1,0.01,0.001,0.0001,0])
10 # create and fit a ridge regression model, testing each alpha
11 model = Ridge()
12 grid = GridSearchCV(estimator=model, param_grid=dict(alpha=alphas))
13 grid.fit(dataset.data, dataset.target)
14 print(grid)
15 # summarize the results of the grid search
16 print(grid.best_score_)
17 print(grid.best_estimator_.alpha)
```

# Random Search

Samples algorithm parameters from a random distribution (uniform) for a fixed number of iterations

Model constructed and evaluated for each combination of parameters chosen

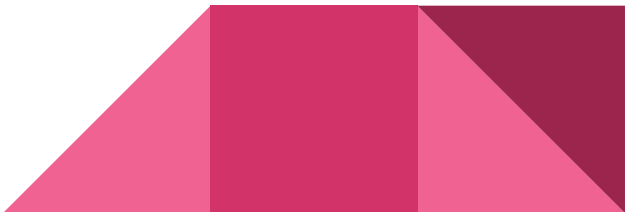


```
1 # Randomized Search for Algorithm Tuning
2 import numpy as np
3 from scipy.stats import uniform as sp_rand
4 from sklearn import datasets
5 from sklearn.linear_model import Ridge
6 from sklearn.model_selection import RandomizedSearchCV
7 # load the diabetes datasets
8 dataset = datasets.load_diabetes()
9 # prepare a uniform distribution to sample for the alpha parameter
10 param_grid = {'alpha': sp_rand()}
11 # create and fit a ridge regression model, testing random alpha values
12 model = Ridge()
13 rsearch = RandomizedSearchCV(estimator=model, param_distributions=param_grid, n_iter=100)
14 rsearch.fit(dataset.data, dataset.target)
15 print(rsearch)
16 # summarize the results of the random parameter search
17 print(rsearch.best_score_)
18 print(rsearch.best_estimator_.alpha)
```

## You can also do it manually...

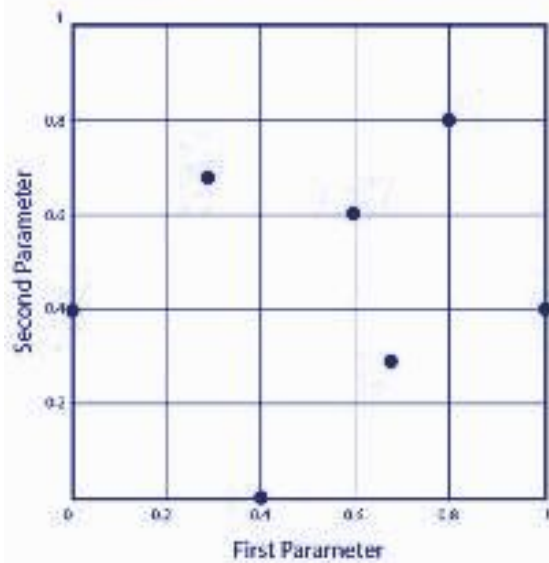
### ALPHA

```
x_values = []
y_values = []
for i in frange (0, 1.1, 0.1):
    classifier_nb_hyper = MultinomialNB(alpha=i)
    classifier_nb_hyper.fit(X_train_bow, train_labels)
    x_values.append(i)
    y_values.append( eval(test_labels,
classifier_nb_hyper.predict(X_test_bow)) )
    plot_curve(x_values, y_values)
```

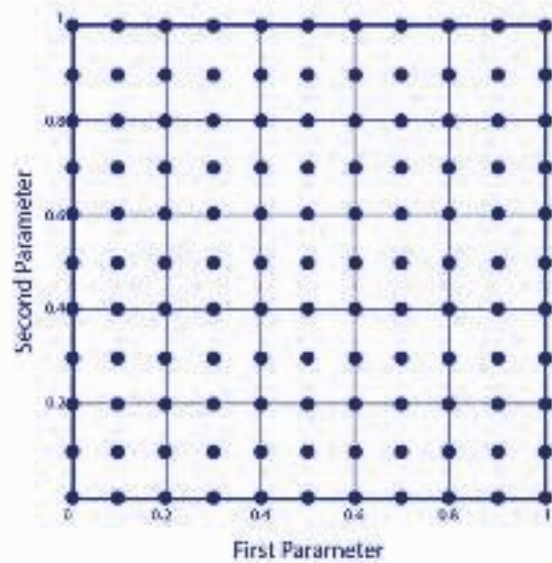




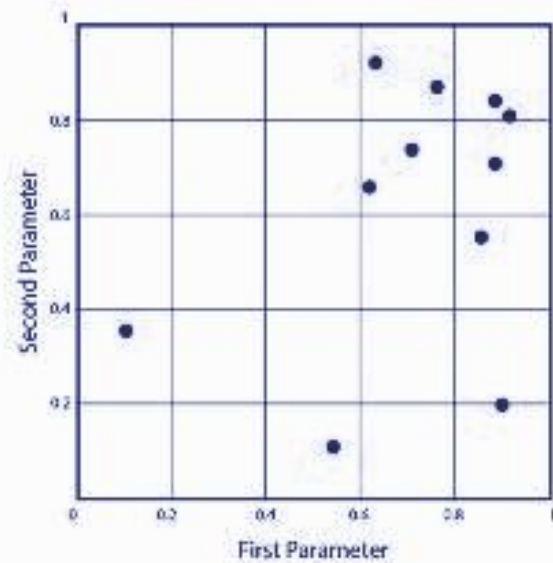
### Manual Search



### Grid Search



### Random Search



# Important Note

- Don't tune your hyperparameters on the testing data!!!
- Testing dataset must always be unseen
- Tune on training or development/validation data

