Lecture 6: Decision Thresholds, Feature Normalization and Selection

David Carlson

Recap

- Last lecture we talked about penalized regression
  - One thing that could be done is *feature selection* using L1 penalties
- We are going to first revisit AUC and decision thresholds to clarify the concept
- We want to be able to do feature selection regardless of what method we're using
  - Hard to put an L1 penalty into k-Nearest Neighbors…

Revisiting Accuracy and AUC

DECISION THRESHOLDS

So far we've talked about two classifiers

- k-Nearest Neighbors:
  - Find the k closest points in space
  - Probability that the new example is positive is given by the fraction of neighbors that are positive
- Logistic Regression
  - Given parameters b, probability of the positive outcome is

$$p(y = positive|x) = \sigma(b \odot x + b_0),$$

$$\sigma(r) = \frac{\exp(r)}{1 + \exp(r)}$$

Decision Threshold

- In classification, we typically end up making a *decision* or *prediction*. Instead of giving the probability of a positive outcome, we make a binary prediction (positive/negative).

- Typically, you would generally guess the *most likely* outcome.
  - If $prob(positive) = .7$ and $prob(negative) =. 3$, then we guess that this will be a *positive* example
  - In binary classification, we choose how confident we need to be to predict a positive outcome.

$$prob(positive) > \tau$$

  - Nominally, τ=.5, but *we get to choose*

Different Types of Errors

	Predict Negative	Predict Positive
True Outcome is Positive	False Negatives (FN)	True Positives (TP)
True Outcome is Negative	True Negatives (TN)	False Positives (FP)

Accuracy=(TP+TN)/All

Changing our decision threshold will change what type of errors we make…

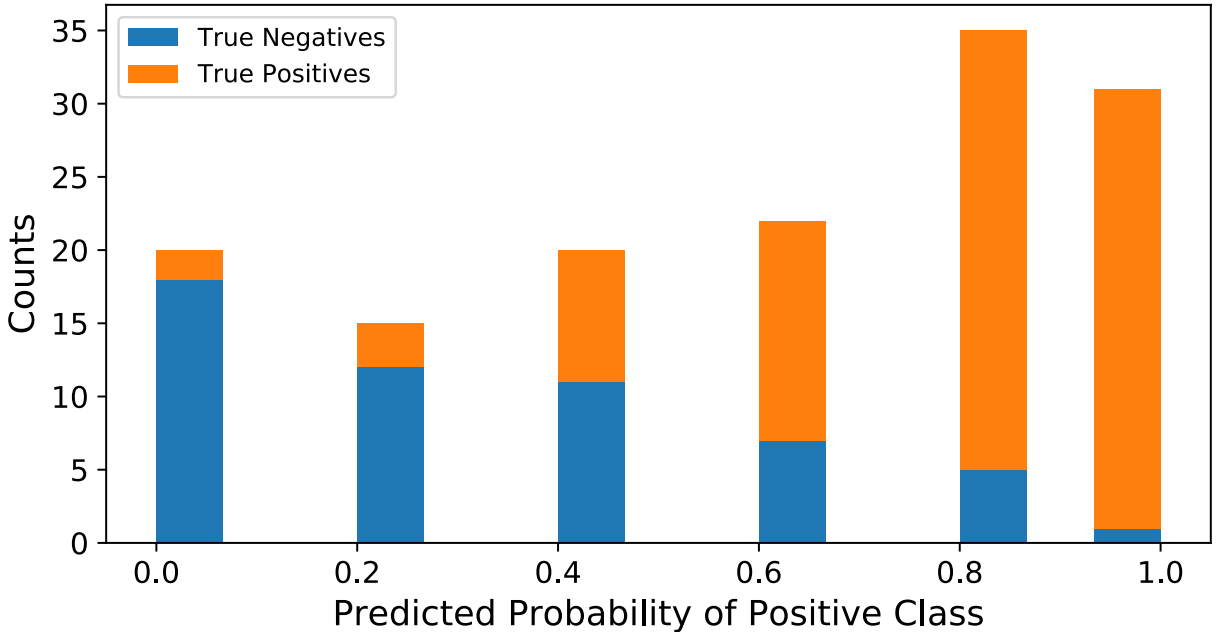
What do the predictions look like on real data?

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	p
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	

This is a famous classic dataset on breast cancer, where the goal is to predict where a tumor is malignant or benign. More details can be found at https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic).
Note: things are much more advanced now…

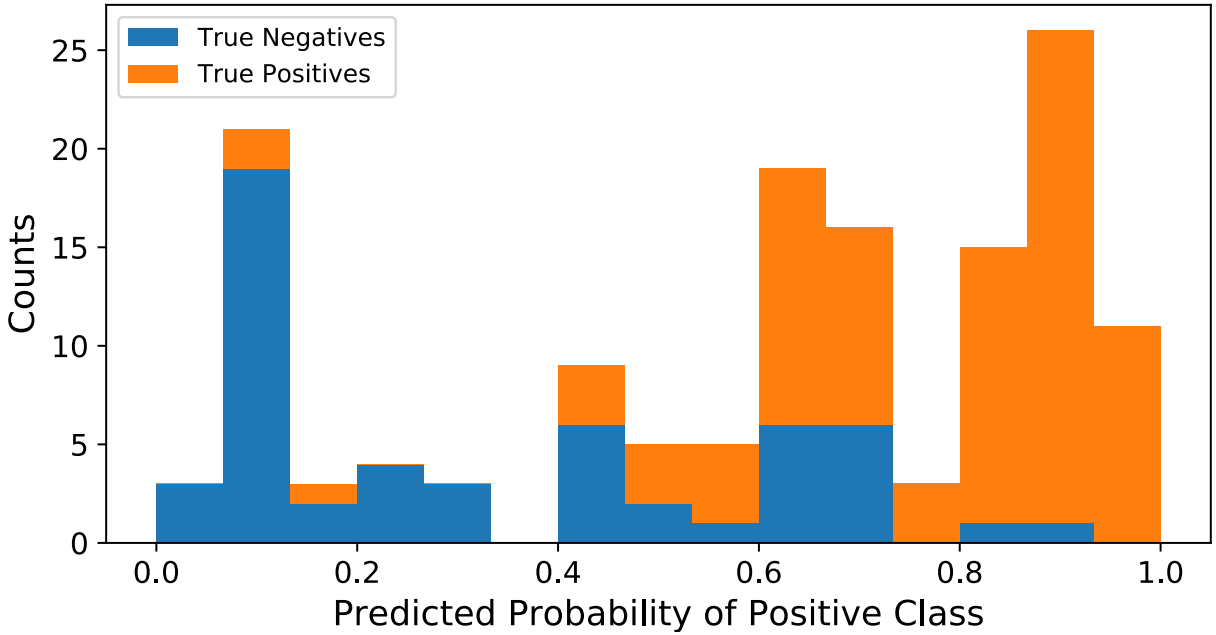
We will go over code to analyze this data in Lecture 8.

Prediction Probabilities on kNN (5 Neighbors)



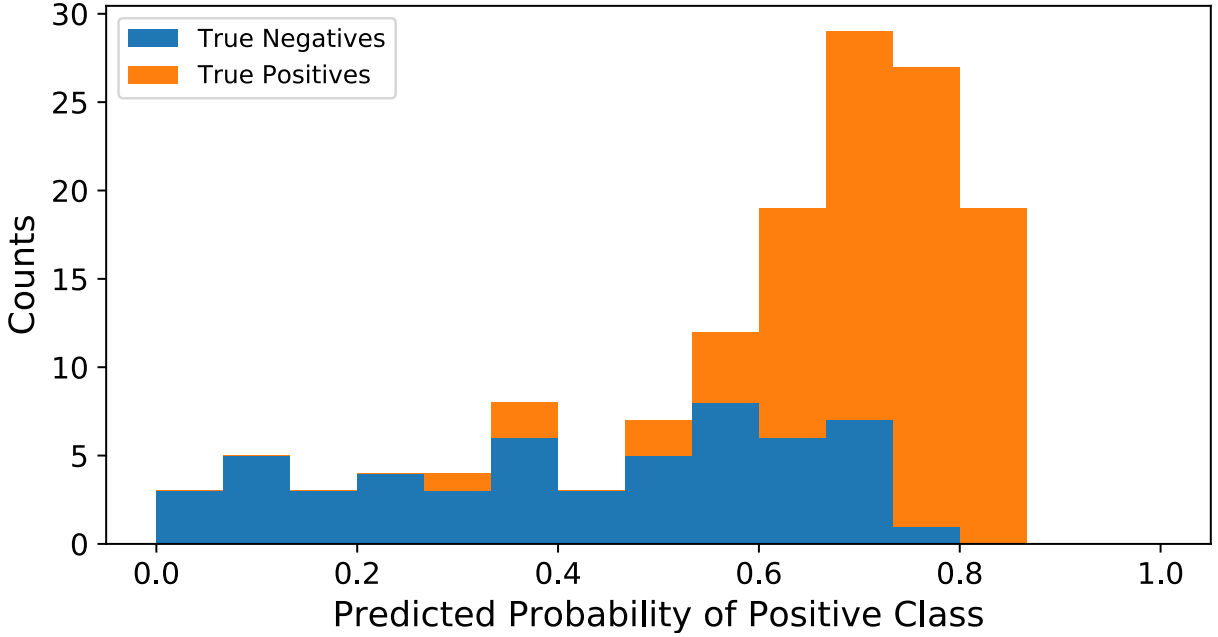
Only 5 neighbors, so only 5 predicted probabilities

Prediction Probabilities on kNN (25 Neighbors)



Using 25 neighbors, so 25 predicted probabilities

Prediction Probabilities from Logistic Regression



Logistic regression can look very different

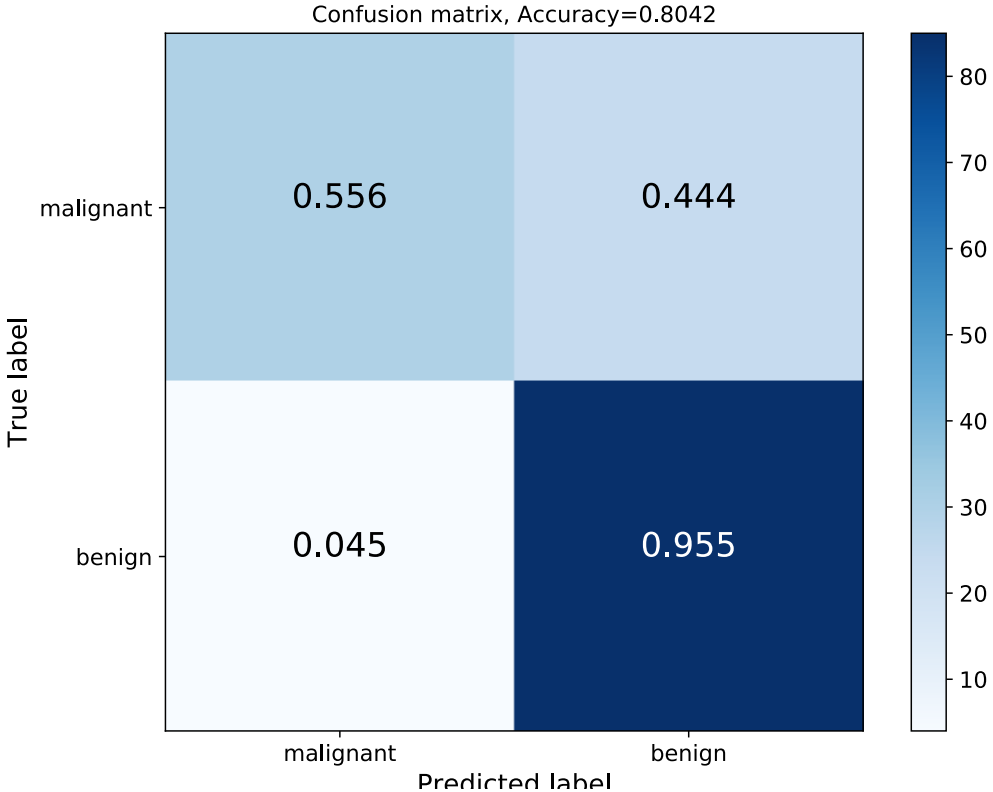
Confusion Matrix

A confusion matrix is one way of visualizing the results of a classification algorithm.

The numbers represent the fraction that a given true class is predicted as that outcome.

The colors represent raw counts.

Here, we are visualizing a binary outcome—this type of visualization easily extends to multi-class problems.

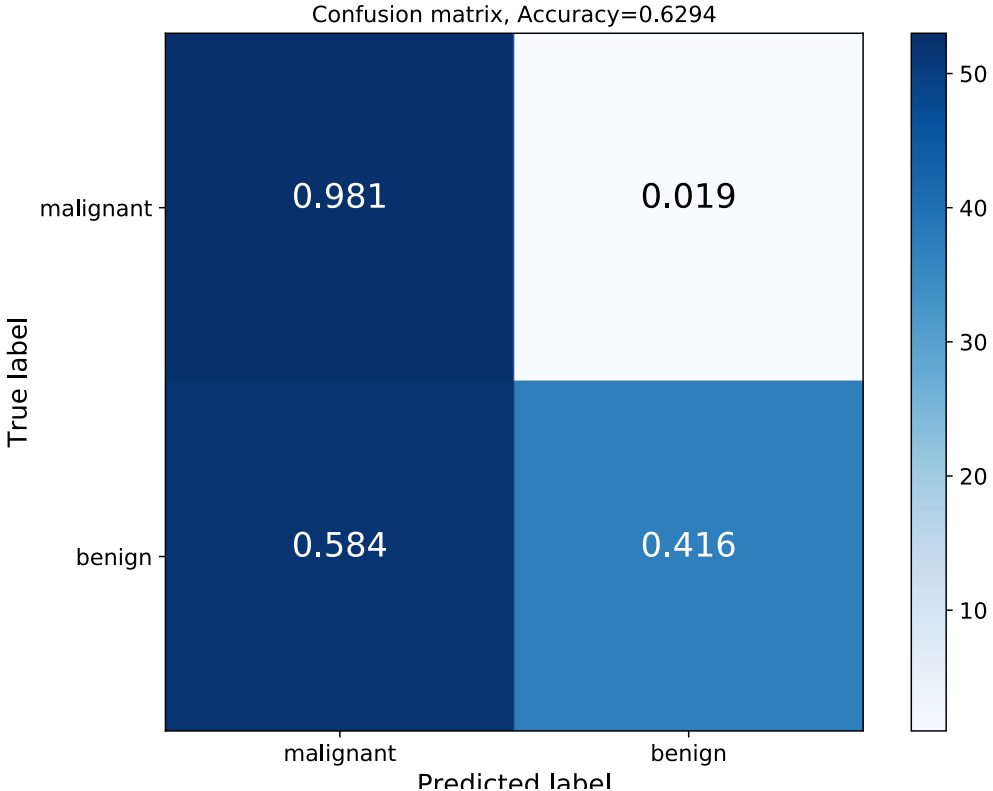


Changing the threshold changes the confusion matrix

Here we are using a much lower threshold for malignant, which drastically changes our properties.

This has great true positive rate, bad false positive rate.

May or may not be better for our actual



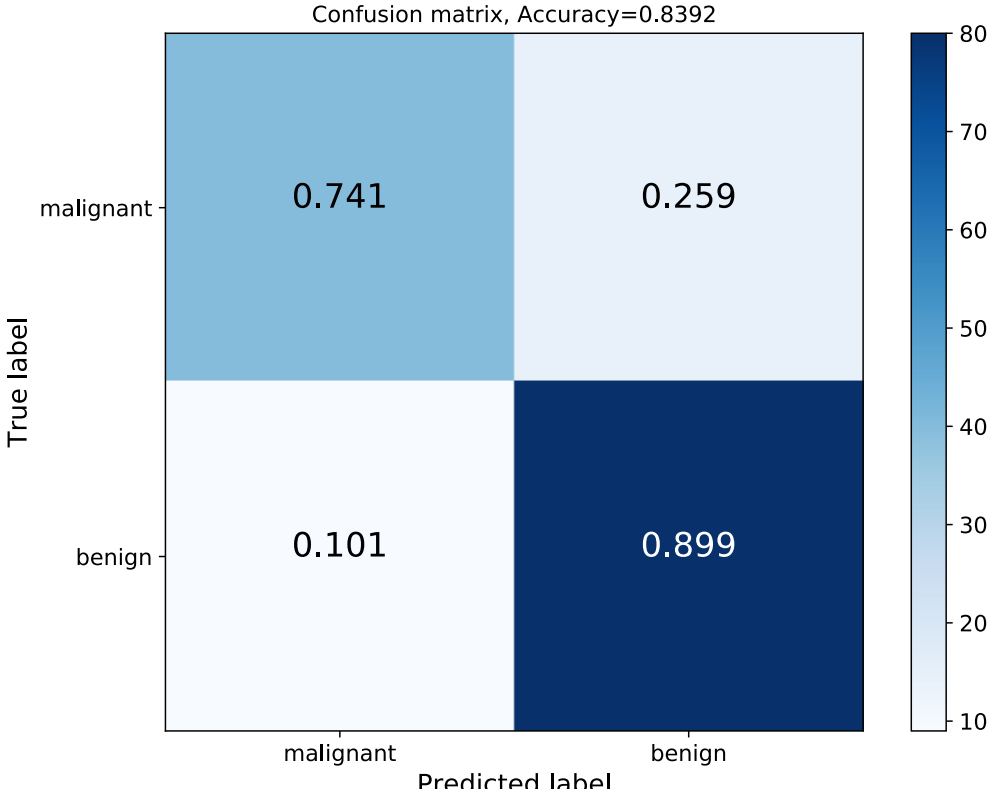
The threshold that maximizes accuracy isn't 0.5

Usually a threshold of 0.4 for a malignant tumor gives better accuracy than 0.5 (the nominal threshold).

This may or may not be a better threshold for our goals.

These are **drastically different** prediction systems with the **same classifier**!

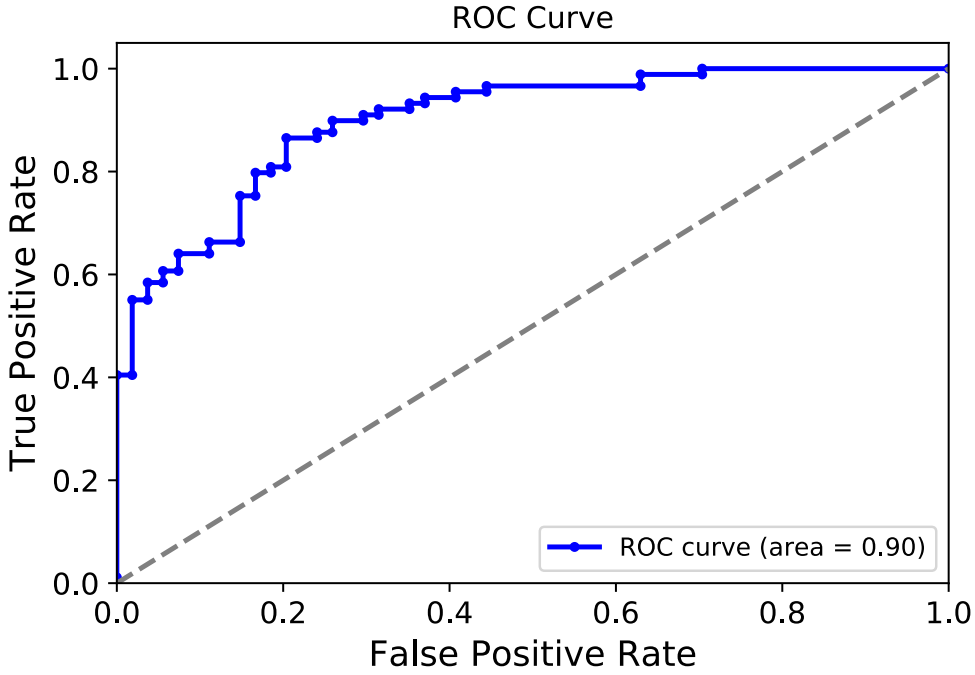
Want to see the result of all thresholds simultaneously.



Receiver Operator Characteristic (ROC) Curve

The usual visualization of the effect of the threshold in a binary problem is the ROC curve.

Each point on the plot is a separate threshold, each with its own confusion matrix.

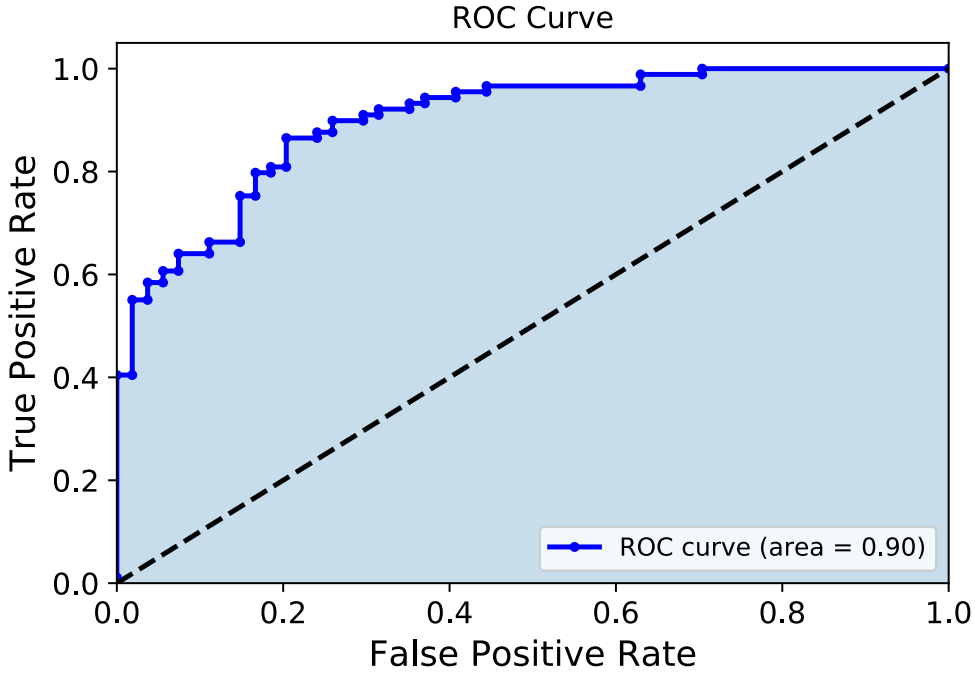


Area Under the Curve

We want a metric that *isn't* dependent on the chosen threshold.

Accuracy is.

One simple solution is to take the area under the ROC curve (AUC).



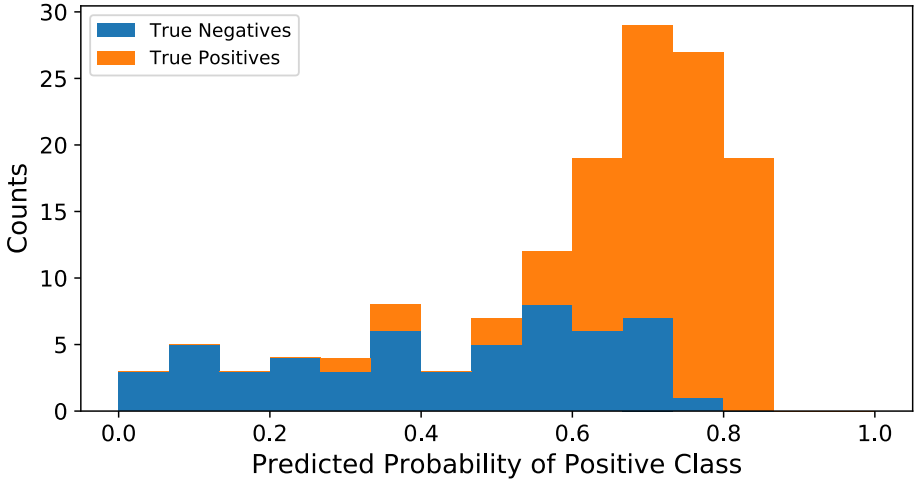
AUC is equivalent to a ranking metric

Consider the following protocol:

- Draw a random sample from the positive class, and get a score (or confidence) r_1 from the classifier (or scoring algorithm)
- Draw a random sample from the negative class, and get a score (or confidence) r_0 from the classifier (or scoring algorithm)
- There is an equivalency here:

$$Prob(r_1 > r_0) = AUC$$

Note that this procedure would be robust to changing the number of samples from either class.

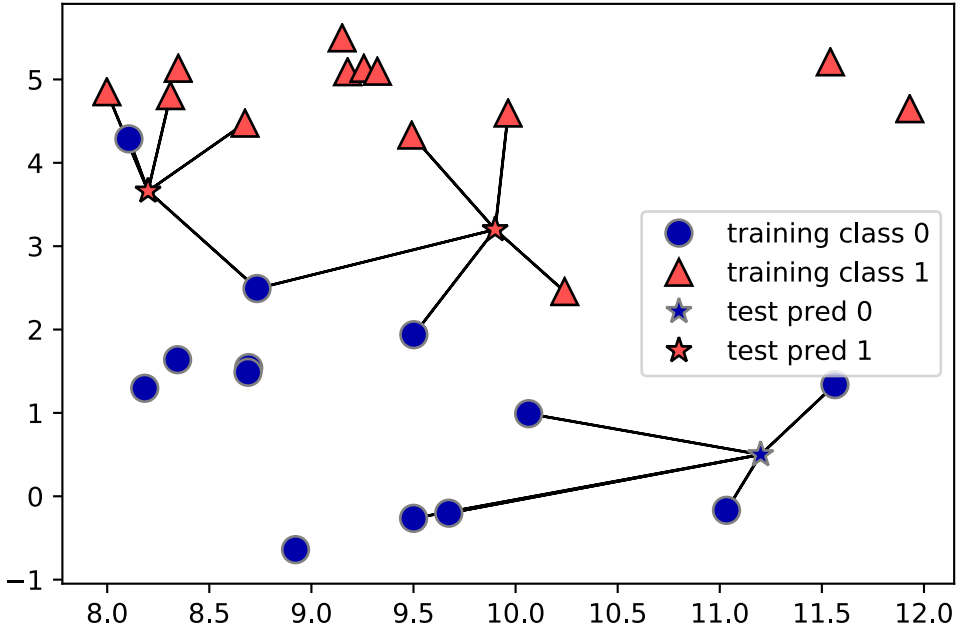


FEATURE SCALING

Overview

- We want to be able to use a subset of the features in k-NN (and later many more methods)
  - Fewer features=simpler model=less overfitting
- Does the scale of features matter?

Revisiting kNN

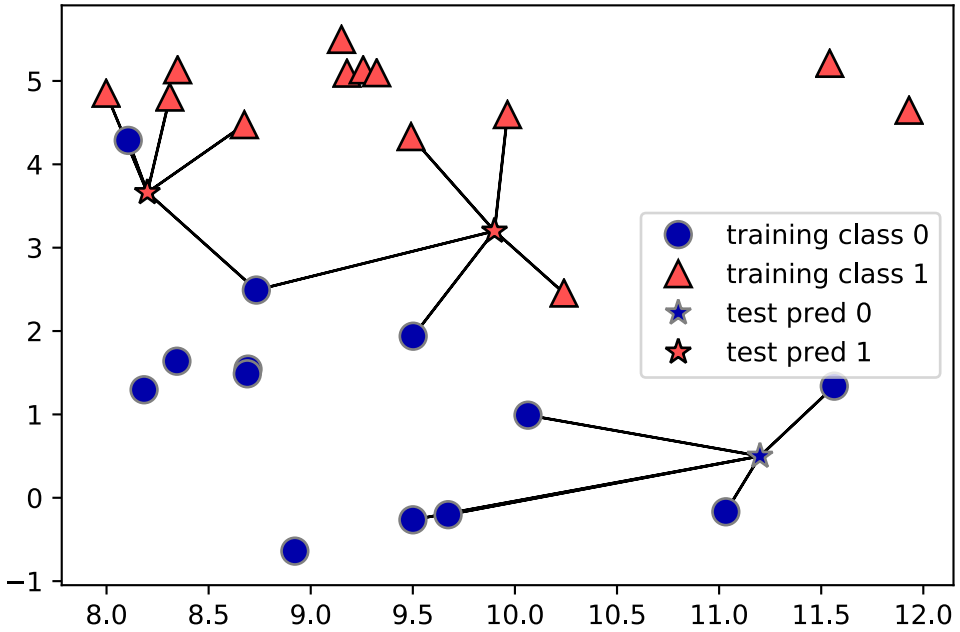


Revisiting kNN

In *ki*-Nearest neighbors, we pick our neighbors based upon who is closest.

Often when we think about distances, we think intuitively about points on a map--the horizontal and vertical dimensions have the same scale and units.

Is this true in data problems?



Considering a real-world dataset

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture	p
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.33	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.41	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.53	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.50	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.67	

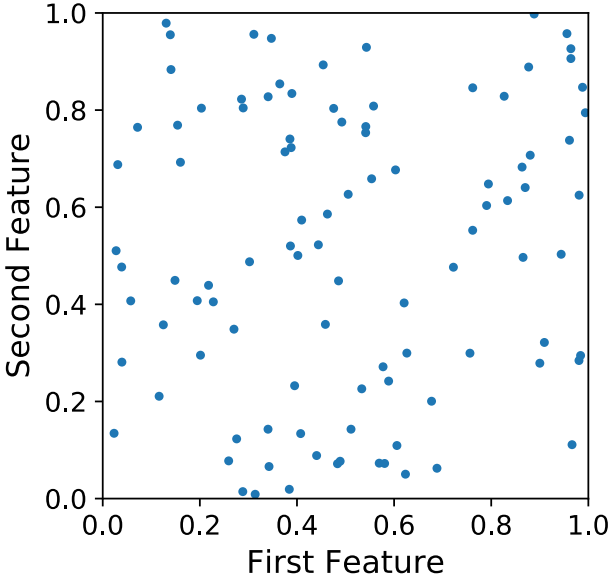
This is a famous classic dataset on breast cancer. More details can be found at https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic).
Note: things are much more advanced now…

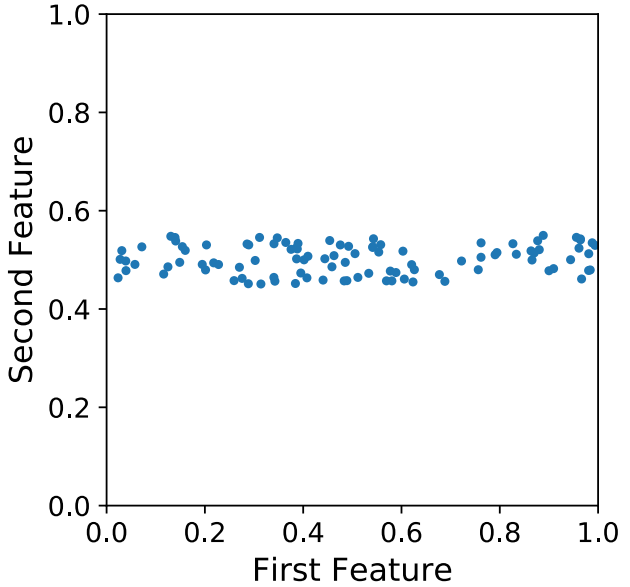
Should the features all be the same?

Feature Normalization

- Why do we need to normalize features?
  - Your neighbors depends on the scale!
  - Tuning parameters are dependent on scaling in many algorithms; implicitly assuming scaling
  - Numerical precision (a feature with comparatively very small numbers can get lost due to rounding errors)
  - Algorithms/models perform better!

Who's your neighbor depends on scale





How to correct this issue?

- Since who is our neighbor (and who is close in kernel space) depends on the scale of the features.
- One common approach:
  - Let's *assume* that all features should have the same summary statistics after normalization
  - This is an *assumption*, so should be considering for how reasonable it is.

- A somewhat contrived counterexample:
  - The first feature is height in feet to the nearest foot.
  - The second feature is the remainder of height (i.e. 5'6" would be [5,6] in this representation)
  - These features should *not* be normalized to have the same statistics

“Z-Scoring”

- Z-scoring means that each feature will be rescaled such that the mean of the feature is 0 and that the standard derivation is 1.

- Let x_{ip} denote the p^{th} feature of the i^{th} sample, and let
  - $\hat{\mu}_p$ be the estimate of the mean for the *p*th feature
  - $\hat{\sigma}_p$ be the estimate of the standard deviation of the *p*th feature

- Then update each feature as:
  - $\tilde{x}_{ip} = \frac{x_{ip} - \hat{\mu}_p}{\hat{\sigma}_p}$
- Note the the resulting distribution is mean 0 with a standard deviation of 1, a “standard” normal

Min-Max Rescaling

- In Min-Max rescaling, each feature is rescaled so that the minimum value for that feature is 0 and the maximum feature is 1.
- Let x_{ip} denote the p^{th} feature of the i^{th} sample, and let
  - $\hat{x}_p^{\max}$ be the estimate of the maximum value of the *p*th feature
  - $\hat{x}_p^{\min}$ be the estimate of the minimum value of the *p*th feature

- Then update each feature as:
  - $\tilde{x}_{ip} = \frac{x_{ip} - \hat{x}_p^{\min}}{\hat{x}_p^{\max} - \hat{x}_p^{\min}}$

Max-Abs Rescaling

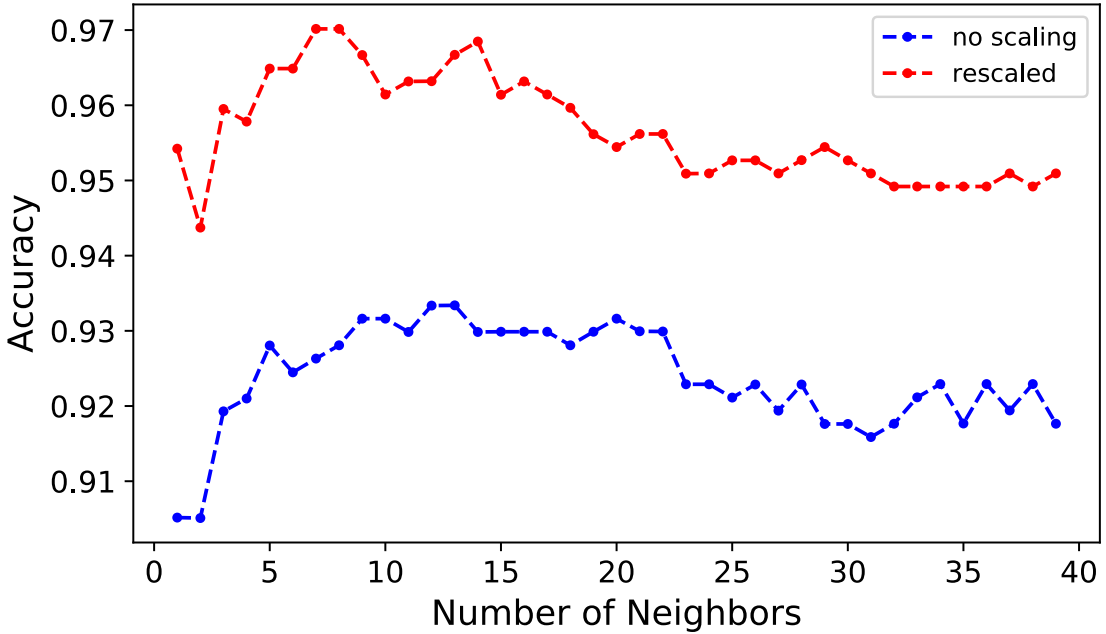
- In Max-Abs rescaling, each feature is rescaled so that the maximum absolute value is 1.
- Let x_{ip} denote the p^{th} feature of the i^{th} sample, and let
  - $|\hat{x}|_p^{\max}$ be the estimate of the maximum absolute value of the *p*th feature
- Then update each feature as:
  - $\tilde{x}_{ip} = \frac{x_{ip}}{|\hat{x}|_p^{\max}}$

Feature Scaling and Cross-Validation

- Feature scaling is an estimated procedure!!
- Must incorporate feature selection into cross-validation.
- Essentially:
  - Estimate quantities necessary for normalization on the training set
  - Apply the transform on the training set
  - For a validation or test set, must apply the *same* transform
- Some comments:
  - Using all data for the normalization seems like a tiny issue, but will overestimate performance in cross-validation
  - Can change the concept of scaling; on a min-max scaling, the test set isn't bounded to 0-1!
- Many other ways of scaling the data

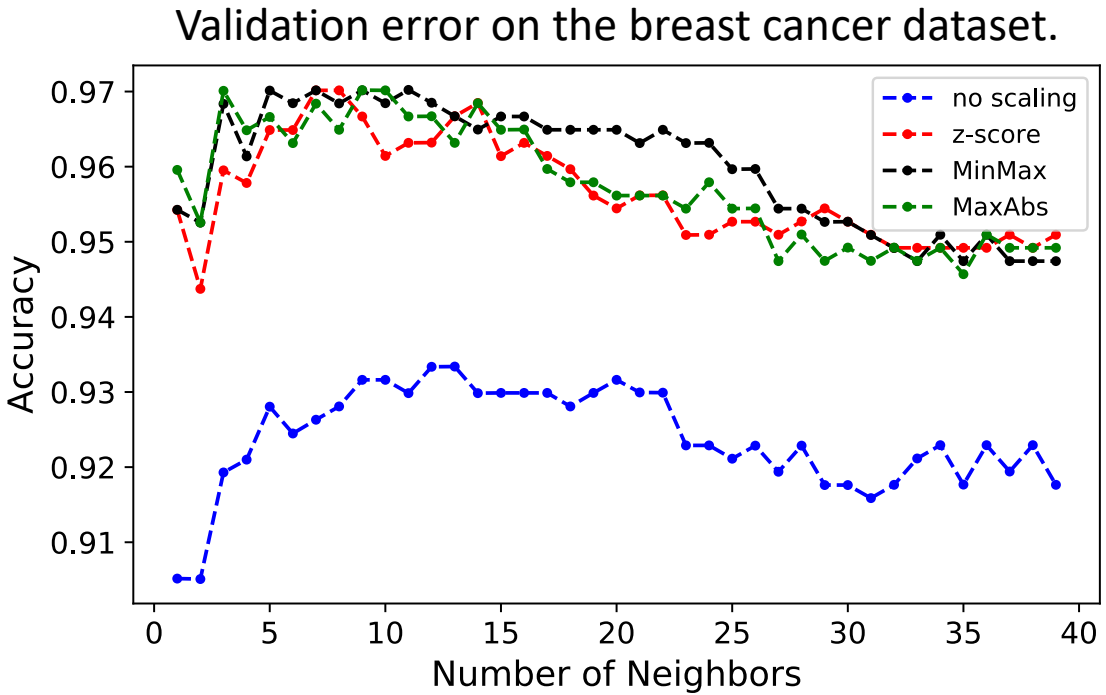
Z-Scoring Makes a Practical Difference

Validation error on the breast cancer dataset.



https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

Different Rescaling Can Affect the Data



https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

Feature scaling also impacts penalization

- If we revisit the cost function for Ridge Regression:

$$\boldsymbol{\beta}^* = \arg\min_{\boldsymbol{\beta}\in\mathbb{R}^p} \sum_{i=1}^{n} \left(y_i - \boldsymbol{\beta}^{\mathrm{T}}\boldsymbol{x}\right)^2 + \lambda\|\boldsymbol{\beta}\|_2^2$$

- Penalty depends on the size of $\boldsymbol{\beta}$
  - How does $\boldsymbol{\beta}$ change as $\boldsymbol{x}$ increases or decreases?

Feature Normalization Summary

- Feature normalization is a critical step that can help improve performance.
- Should almost always be included
- Sometimes, one normalization will be *much* better than another–requires understanding and knowing your data!
- Usually, which one doesn't matter too much. Default is a standard scaler or Z-score.

How can we choose a subset of features to improve our classifier?

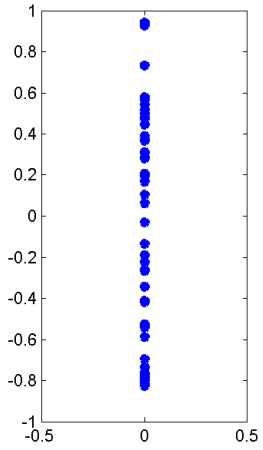
SUBSET SELECTION

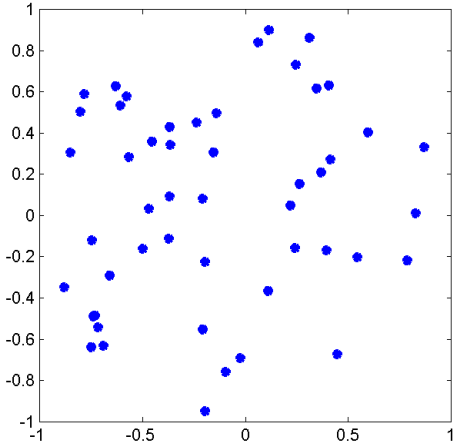
Why Feature Selection?

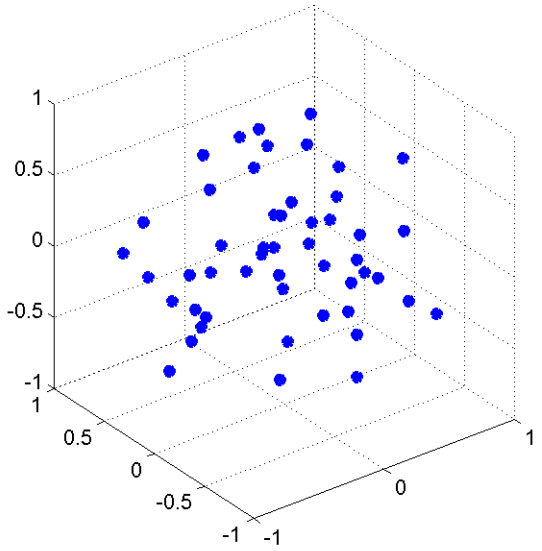
- There are two main reasons why we are often not satisfied with using all features.
- Prediction Accuracy
  - Using all features requires a complex model that can overfit
  - Removing features can *bias* estimates (we implicitly state that all features not in the model are not related to the outcome) but reduces variance
- Interpretation
  - With a small number of features, it is easier to determine what is important to the predictions

Curse of Dimensionality

- 50 data points, all within 1 unit of the origin (L_2-norm)



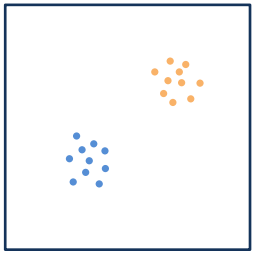




Desirable Feature Properties

MAXIMIZE RELEVANCY

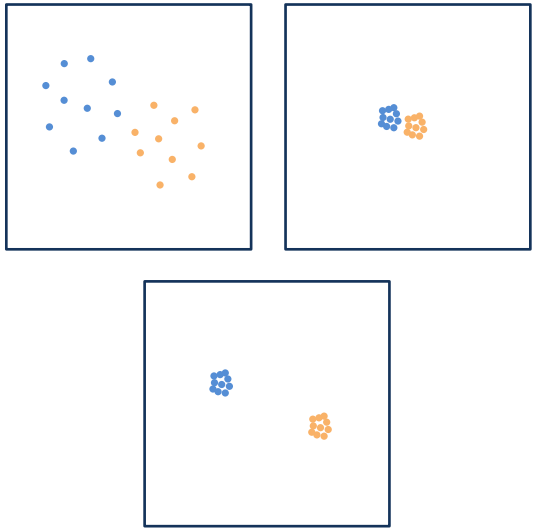
MINIMIZE REDUNDANCY



Feature Selection Philosophies

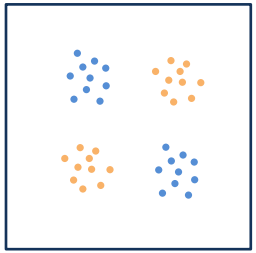
CLASSIFIER AGNOSTIC METHODS

Evaluate the class separability of the candidate feature sets independent of the classifier



CLASSIFIER SPECIFIC METHODS

Evaluate classifier performance with the candidate feature sets

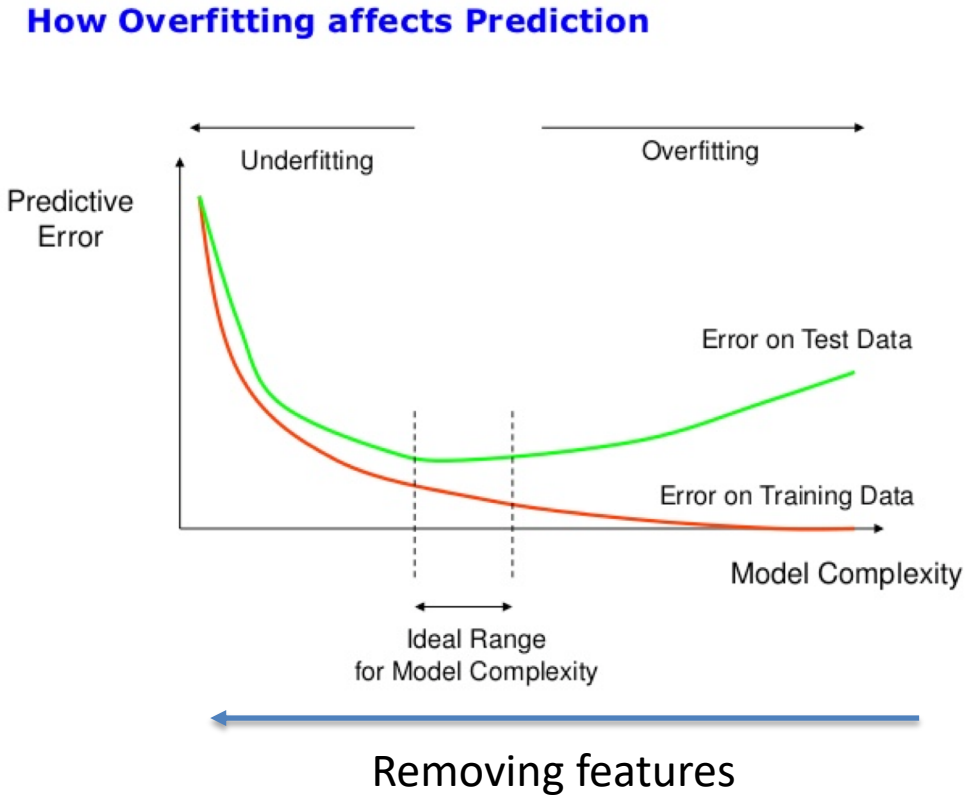


Overfitting

If we choose relevant features, we can potentially reduce complexity, reduce overfitting, and improve performance.

Using more features leads to a more “complex” model. With an abundance of data, we want to use all features.

In practice, data is limited (even ”big” datasets need to control model complexity).



Best Subset Selection

- Let's find the best feature representation:
  - Set the number of features used $k \in \{1, \ldots, p\}$, assuming p total features
  - For each value k, try the following:
    - Try every possible combinations of k features, and evaluate performance with cross-validation.
  - Pick the value of k and the set of features that gives the best performance
  - Call this the best set, hope it gives the best performance

- Is this approach good?

Comments on “Best Subset Selection”

- Can be computationally feasible up to 30-40 features with clever tricks in certain algorithm classes.
  - Nearest neighbors this is essentially impossible for more than 10-15 dimensions
- Requires too much trial-and-error, try too many cross-validation estimates (ends up with biased results upwards)
- In classical linear models, can use Ridge or Lasso regression, but those methods aren’t general

Univariate Statistics

- Instead of trying to figure out how the features work *together*, let's evaluate each feature on its own.
- How can we determine the relationship between each individual feature?
- One common approach is to use the "F-score," which is essentially a one-way ANOVA on a single variable. This is given by:
- $F = \frac{explained\ variance}{unexplained\ variance} = \frac{n_0(\bar{x}_0 - \bar{x})^2 + n_1(\bar{x}_1 - \bar{x})^2}{(n_0 - 1)\sigma_0^2 + (n_1 - 1)\sigma_1^2}$
- 0 denotes the "negative" class, 1 denotes the "positive" class
- Essentially, this asks whether the statistics are different if we consider two groups (i.e. the 0 group and the 1 group) compared to combining the groups.
- Going to skip the derivation.

Applying the univariate feature selection

- Many possible rules for feature selection:
  - Keep a % of features with the highest classification rates
  - Define a significance threshold, and only keep features that pass that threshold
  - Etc.

- Seems simple, are there any downsides?
  - Does *not* account for correlations between features
  - Does *not* account for interactions between features

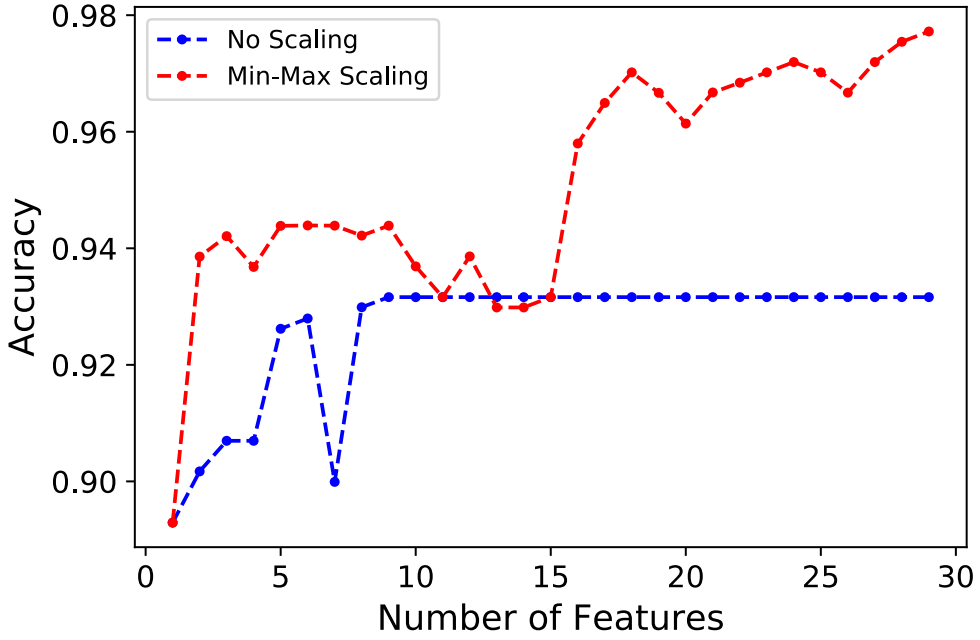
- Regardless of these drawbacks, often quite good in practice.

Feature Selection in Practice

We can apply this feature selection to the same dataset on kNN with 10 neighbors.

Here, we *probably* want to use all features.

Rescaling still matters!



Many other choices for univariate statistics

- Pearson correlation (or other correlation metrics)
- Chi-Squared statistics (still correlation)
- Mutual Information
- Run a classifier on each feature
- Total Variance (throw out low-variance features)

What about sequential methods?

- Instead of choosing the features all at once, we can examine the strategies that would happen if we choose features sequentially?
- Why would we want to choose features sequentially?
  - Features can be highly correlated. If two features are essentially the same, they don't add much to the classification, and effort should be focused on other features.
  - Features can have non-linear interactions. If two features interact in a highly nonlinear manner, its important that the classifier estimate how these interaction terms affect classification.
- Sequential methods are typically *model-dependent*, meaning that the chosen model affects which features are selected.
  - Different models may select different features…

Forward-Stepwise Selection

- Consider the following procedure:
  - Start with 0 features
  - While the cross-validation performance continues to improve:
    - 1. Try adding one feature from all possible features not currently in the model
    - 2. Add the feature that tests the best if it improves the classification performance
- This procedure adds features in a “greedy” manner.

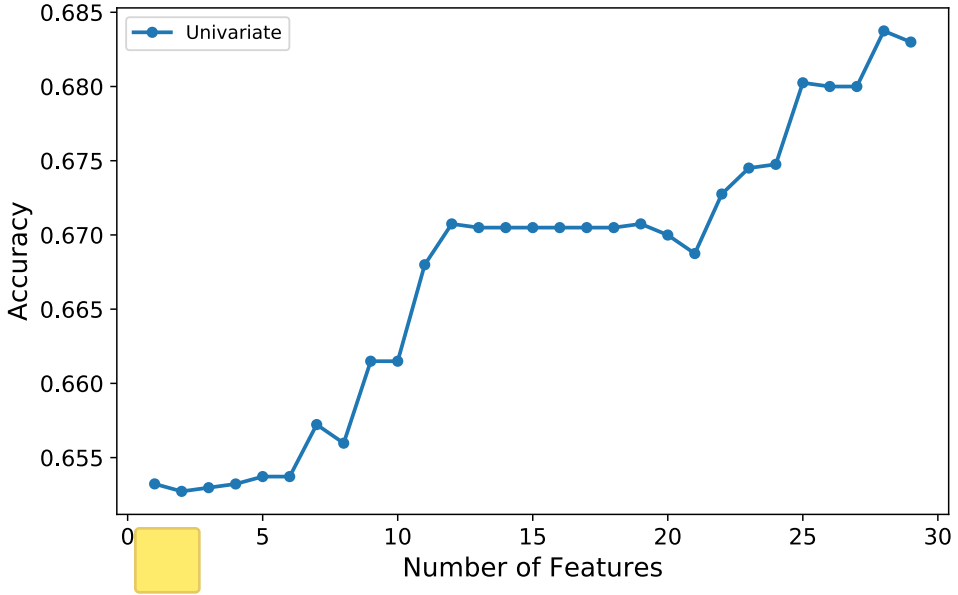
Let’s apply these techniques on a slightly harder dataset

- We’re going to use the “Forest Cover” dataset
- Goal is to predict major species of tree in a plot of land (e.g douglas fir or cottonwood)
- http://archive.ics.uci.edu/ml/datasets/Covertype

Univariate Feature Selection

Univariate feature selection can be quite positive here.

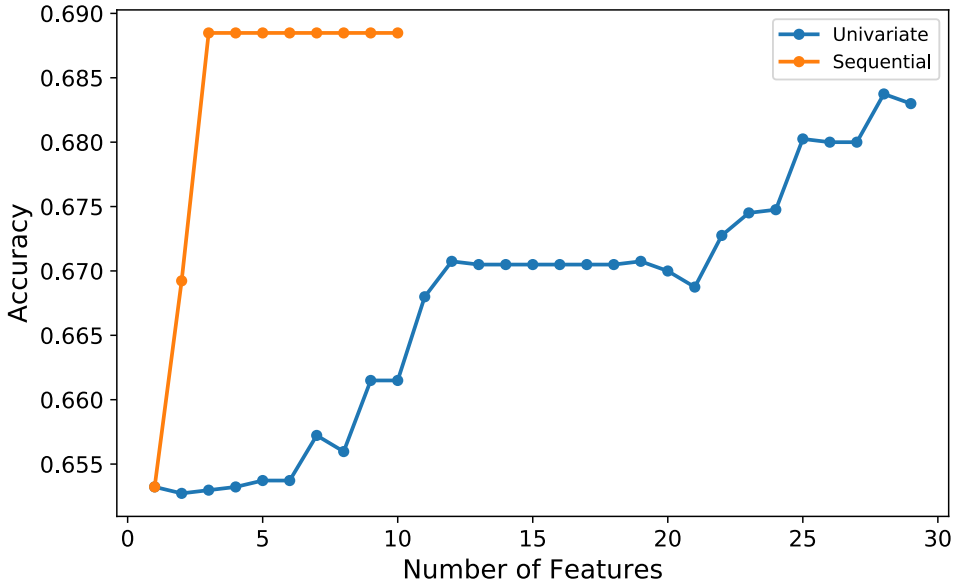
Seems highly encouraging, and appears that we want to use all features.



Forward-Stepwise Tells a Different Story

Univariate feature selection can be quite positive here.

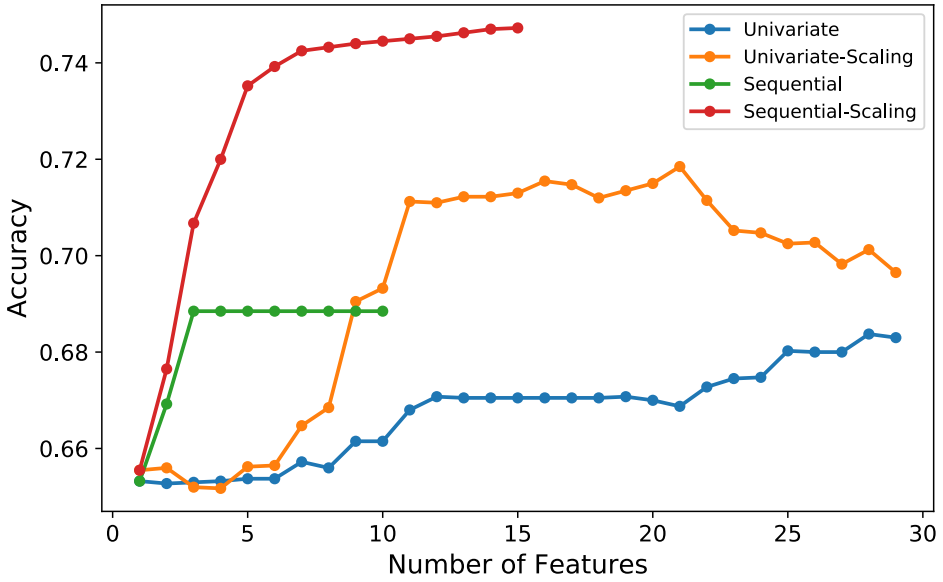
Forward Stepwise implies that we only need *3* features…



Still need to consider rescaling the features

The only change here is that the features are scaled prior to analysis pipeline.

Rescaling makes us get rather different properties!



Backwards-Stepwise Selection

- Consider the following procedure:
  - Start with all features in the model
  - While the cross-validation performance continues to improve:
    - 1. Try removing one features currently in the model
    - 2. Remove the feature that tests the best if it improves/doesn't hurt the classification performance
- This procedure removes features in a "greedy" manner.
- Often works very well in practice also.

Conclusions

- Feature scaling is often critical for performance
  - Must make sense in the context of the data! Rescaling shouldn't be data-agnostic
- Feature selection can greatly enhance our machine learning methods
- Both of these techniques should be included in most analysis techniques
- Forward-stepwise regression is a very standard technique in scientific analysis.
  - Fewer features are more interpretable!