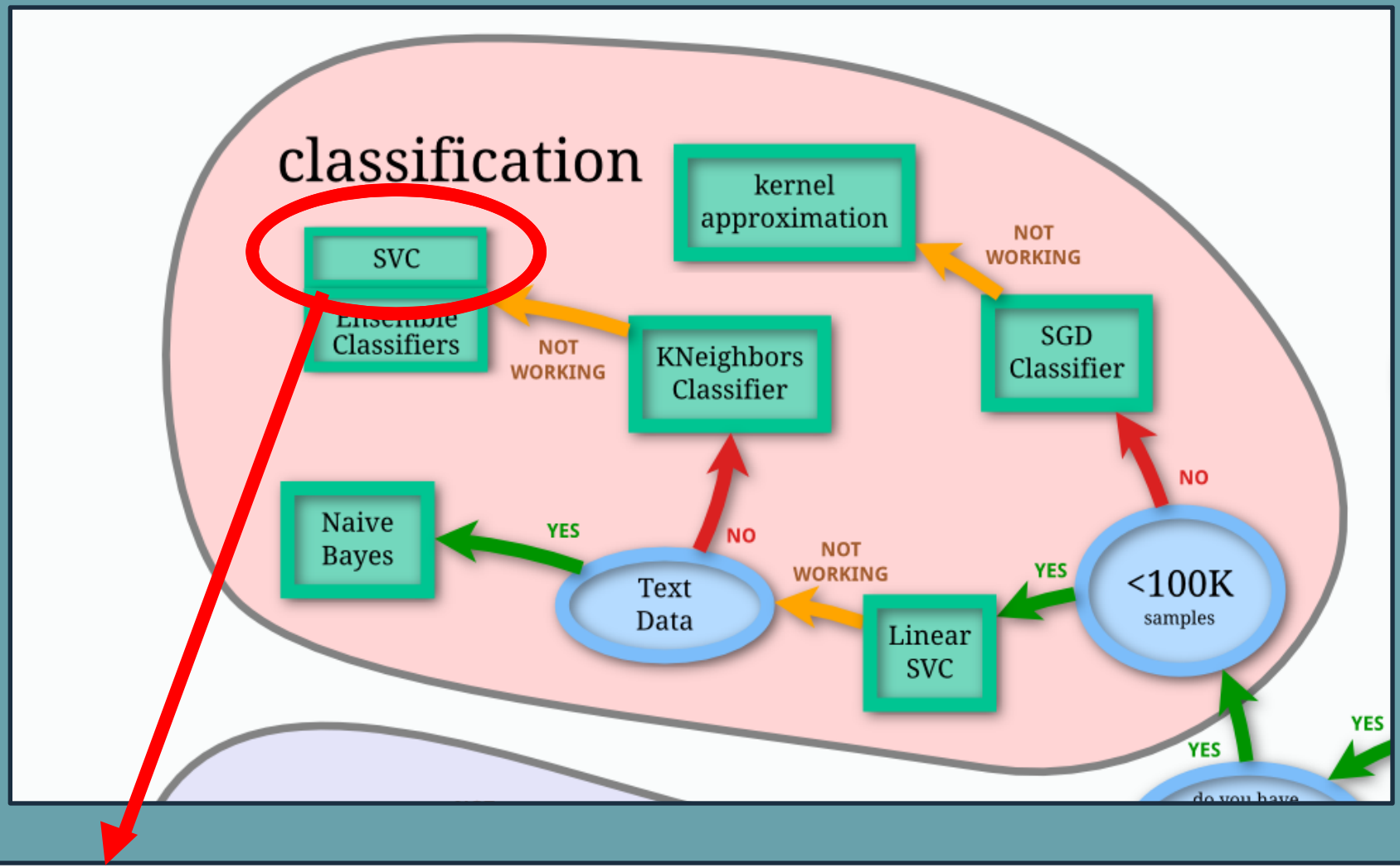
OPTIMIZATION AND EVALUATION OF CLASSIFIERS

Callie Hood, ASTR 503



class `sklearn.svm.` **SVC** (*C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape=None, random_state=None*) ¶ [source]

- How do I judge the performance of my classifier?
- How do I pick the best hyperparameters to optimize my classifier's performance?

Step 1: Pick a performance metric.

The Accuracy Paradox

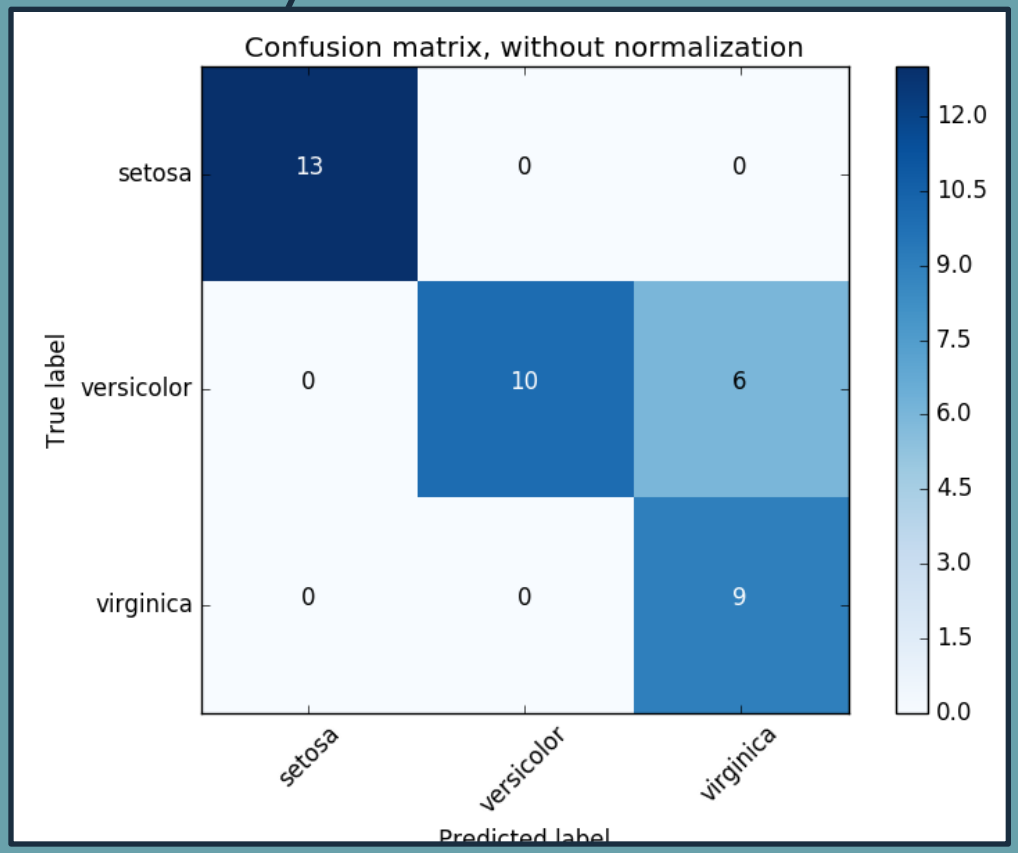
- Scikit-learn's default scoring metric for classifiers is accuracy, or

$$\frac{\textit{Number of correct predictions}}{\textit{Number of total predictions}} = \frac{TP + TN}{TP + TN + FP + FN}$$

- For binary classification, we can define:
  - ***True Positives (TP):*** *number of positive examples, labeled as such.*
  - ***False Positives (FP):*** *number of negative examples, labeled as positive.*
  - ***True Negatives (TN):*** *number of negative examples, labeled as such.*
  - ***False Negatives (FN):*** *number of positive examples, labeled as negative.*
- When TP < FP, the accuracy will always increase when we assign "negative" to every input data point. Conversely, when TN < FN, the same will happen when we label everything as "positive." The *accuracy paradox* refers to the problem when we can change our classifier to a completely useless model with zero predictive ability, yet the accuracy increases

Confusion Matrix

- A *confusion matrix* has the predicted labels on the x-axis and the true labels on the y-axis



- Correct predictions lie on the diagonals of the matrix
- Sklearn's confustion_matrix function will compute each entry for you

Precision & Recall

- Two possibly better metrics are precision and recall
- Precision: percentage of correct positive classifications (i.e. resistance to contamination)

$$P = TP / (TP + FP)$$

- Recall: percentage of instances from the positive class that were identified correctly (i.e. completeness)

$$R = TP / (TP + FN)$$

- Precision and Recall are often in opposition with one another- thus, may want to look at both simultaneously with the F1-Metric, the harmonic mean of precision and recall:

$$F_1 = \frac{2 \times \text{precision} \times \text{recall}}{(\text{precision} + \text{recall})}$$

Scoring Metrics in Scikit Learn

- Sklearn's classification_report function will compute the precision, recall, and F1 score for each class

```
>>> from sklearn.metrics import classification_report
>>> y_true = [0, 1, 2, 2, 2]
>>> y_pred = [0, 0, 2, 2, 1]
>>> target_names = ['class 0', 'class 1', 'class 2']
>>> print(classification_report(y_true, y_pred, target_names=target_names))
             precision    recall  f1-score   support

    class 0       0.50      1.00      0.67         1
    class 1       0.00      0.00      0.00         1
    class 2       1.00      0.67      0.80         3

avg / total       0.70      0.60      0.61         5
```

- These are just some of the most common metrics- many more can be found in the sklearn.metrics module
  - *Different requirements/uses for each function*
  - *Many control the weighted contribution of each sample to the overall score with the sample_weight parameter, allowing one way to correct for imbalanced data*

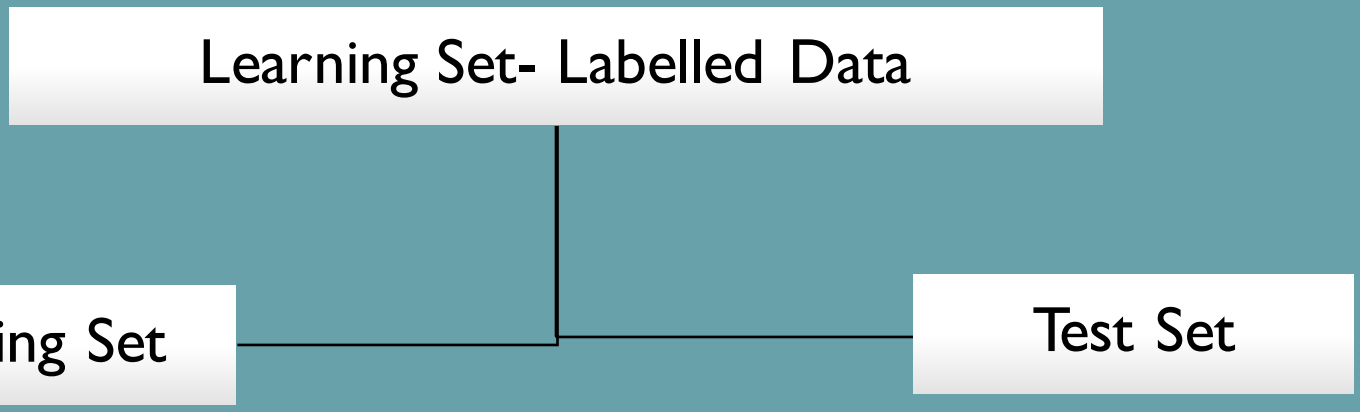
Scoring Metrics in Scikit Learn

- You can also use the sklearn function make_scorer to create your own customized scorer from a simple python function to be used in conjunction with other sklearn functions and algorithms
- Key to pick a performance metric that is tied to your science goals- what are you doing with your classifications?
  - If you're using your classifications to compute some other parameter, you could define your own scorer that minimizes the error on your derived parameter (i.e. maybe as some combination of precision and recall)
  - You may care more about minimizing errors in some classes than others, or in particular directions (i.e. minimizing false negatives at the expense of a higher false positive rate)

Step 2: Optimize your classifier.

Optimization Procedure

- Most machine learning algorithms have some number of hyperparameters that need to be optimized in order to give you the best performance
- To start, you need to define your training and test sets.



- Used to find optimal hyperparameters via k-fold cross validation and to train your algorithm

- Used to evaluate your classifier's performance on "new" data (data that has not been used to train your algorithm)

Parameter Grid Search

- Sklearn's GridSearchCV and RandomizedSearchCV perform cross-validated grid searches over a parameter grid to identify the best combinations of the parameters given

```
In [38]: # create a parameter grid: map the parameter names to the values
          that should be searched
         param_grid = dict(n_neighbors=k_range)
         print param_grid

         {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
          15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 3
         0]}

In [39]: # instantiate the grid
         grid = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy')
```

```
In [40]: # fit the grid with data
         grid.fit(X, y)
```

```
In [45]: # examine the best model
         print grid.best_score_
         print grid.best_params_
         print grid.best_estimator_

         0.98
         {'n_neighbors': 13}
         KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='min
         kowski',
                     metric_params=None, n_jobs=1, n_neighbors=13, p=2,
                     weights='uniform')
```

Final Evaluation of Classifier

- The optimized grid.best_score_ estimate *may* be a biased estimate of the true performance of the model since the model's construction was guided by the optimization of this quantity
- You can calculate the score for your optimized classifier using the test set that was left out of the parameter search
- Alternatively, you can use **nested** cross-validation for correctly selecting the model **and** correctly evaluating its performance with your entire learning set- because of the cross-validation, your classifier is not scored on the same data used to train it

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

scores = cross_val_score(
            GridSearchCV(KNeighborsClassifier(),
                         param_grid={"n_neighbors": list(range(1, 100))},
                         scoring="accuracy",
                         cv=5, n_jobs=-1),
            X, y, cv=5, scoring="accuracy")

# Unbiased estimate of the accuracy
print("%f +-%f" % (1. - np.mean(scores), np.std(scores)))
```

```
0.144000 +-0.023958
```

Optimization beyond hyperparameters

- Depending on your science goals, choice of classifier, and data there may be other ways to tune your classifier's output
  - Feature Selection: sklearn.featureselection offers many different approaches for determining the most important features in your dataset.
  - Scaling: Scaling features is critical to some classifiers (i.e. SVCs). scikitlearn.preprocessing contains multiple different scaling algorithms that could affect your classifications differently
  - Oversampling/undersampling: With imbalanced data, rather than altering the cost of misclassification of different classes you can also sample your data differently to create a more balanced training set
    - *Oversampling: add copies of instances from the under-represented class*
    - *Undersampling: delete instances from the over-represented class*

Summary and Conclusions

- Although scikit-learn offers many out-of-the-box algorithms to apply to your classification problem, **human input** is key to your success
  - The definition of success for your classifier depends on what you are doing with your classifications.
  - There is no one best metric for evaluating your classifier- instead, different metrics will give you alternative viewpoints on your classifier's performance.
  - Many classification algorithms require fine-tuning of their parameters for optimal classifications. Many sklearn functions exist to ease this process.

References

- http://scikit-learn.org/stable/modules/model_evaluation.html
- http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py
- http://machinelearningmastery.com/how-to-evaluate-machine-learning-algorithms/
- http://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python/
- Example iPython notebooks:
  - *https://app.dominodatalab.com/u/marks/scikit-learn-examples/view/08_grid_search.ipynb?commitId=aeb66164 1f676c84fa46f4339 4dfb16408ee8ddb*
  - *https://github.com/glouppe/tutorials-scikit-learn/blob/master/1.%20An%20introduction%20to%20Machine%20Learning%20with%20Scikit-Learn.ipynb*