# INFO 284 – Machine Learning

## Machine Learning Practice

Bjørnar Tessem & Marija Slavkovik

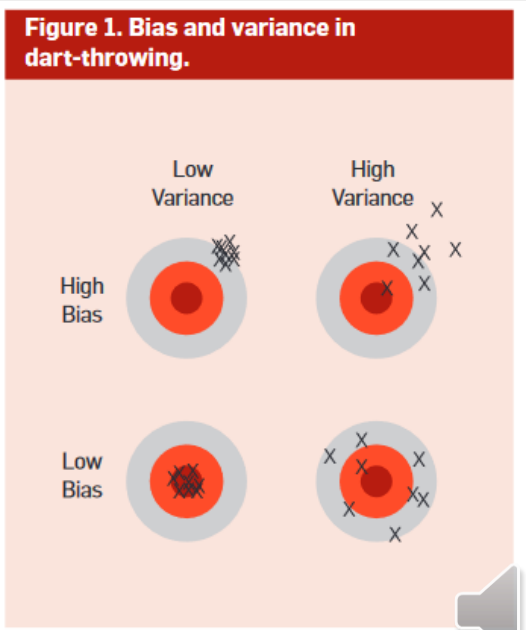
UNIVERSITY OF BERGEN

# A few useful things to know about machine learning

- Learning = Representation + Evaluation + Optimization
- **Representation** – Classifier's hypothesis space, How to represent input
- **Evaluation** – objective function, loss function, cost function
- **Optimization** – Find the optimal parameters for the classifier

# A few useful things to know about machine learning

- It is generalization that counts
- Data alone is not enough
- Overfitting has many faces
- Intuition fails in high dimensions
- Theoretical guarantees are not what they seem



# A few useful things to know about machine learning

- More data beats a clear algorithm
- Learn many models not just one
- Simplicity does not imply accuracy
- Representation does not imply learnable
- Correlation does not imply causation

# Representation - Feature Engineering

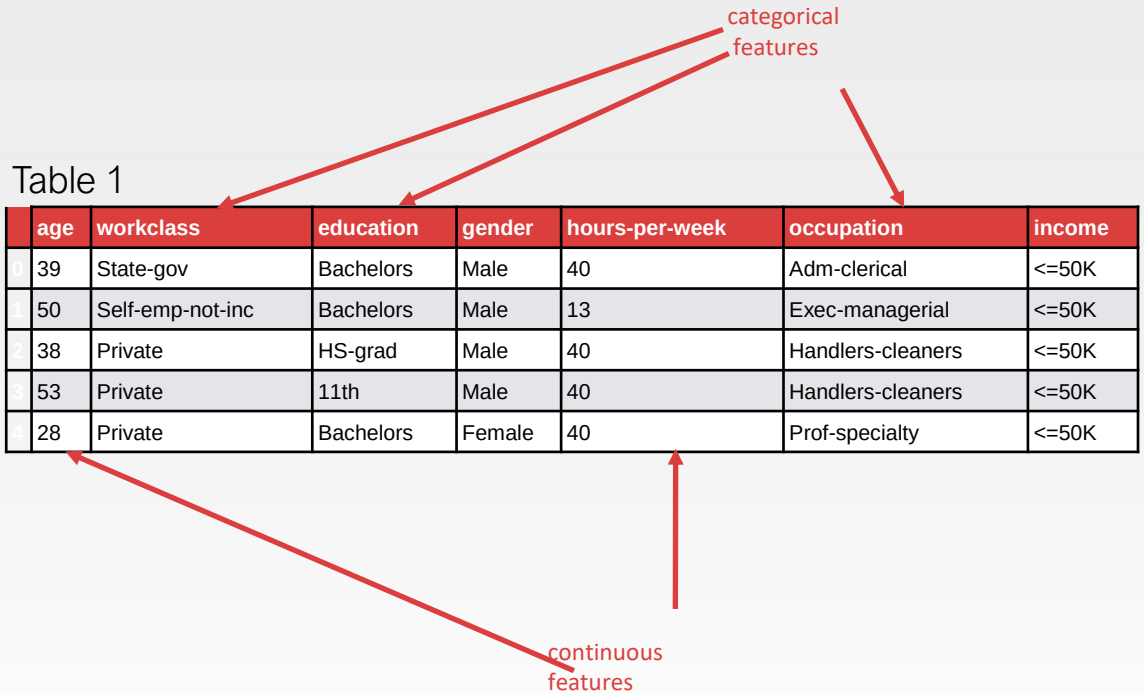



Table 1

| | age | workclass | education | gender | hours-per-week | occupation | income |
|---|---|---|---|---|---|---|---|
| 0 | 39 | State-gov | Bachelors | Male | 40 | Adm-clerical | <=50K |
| 1 | 50 | Self-emp-not-inc | Bachelors | Male | 13 | Exec-managerial | <=50K |
| 2 | 38 | Private | HS-grad | Male | 40 | Handlers-cleaners | <=50K |
| 3 | 53 | Private | 11th | Male | 40 | Handlers-cleaners | <=50K |
| 4 | 28 | Private | Bachelors | Female | 40 | Prof-specialty | <=50K |

$$\hat{y} = w[0] * x[0] + w[1] * x[1] + \cdots + w[p] * x[p] + b$$

# One-hot-encoding

- Every categorical feature that admits n categorical values is represented with n binary valued features

| | age | workclass | education | gender | hours-per-week | occupation | income |
|---|---|---|---|---|---|---|---|
| **0** | 39 | State-gov | Bachelors | Male | 40 | Adm-clerical | <=50K |
| **1** | 50 | Self-emp-not-inc | Bachelors | Male | 13 | Exec-managerial | <=50K |
| **2** | 38 | Private | HS-grad | Male | 40 | Handlers-cleaners | <=50K |
| **3** | 53 | Private | 11th | Male | 40 | Handlers-cleaners | <=50K |
| **4** | 28 | Private | Bachelors | Female | 40 | Prof-specialty | <=50K |

| | age | workclass_ state-gov | workclass_ self | workclass_ private | education | gender | hours-per-week | occupation | income |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 39 | 1 | 0 | 0 | Bachelors | Male | 40 | Adm-clerical | <=50K |
| **1** | 50 | 0 | 1 | 0 | Bachelors | Male | 13 | Exec-managerial | <=50K |
| **2** | 38 | 0 | 0 | 1 | HS-grad | Male | 40 | Handlers-cleaners | <=50K |
| **3** | 53 | 0 | 0 | 1 | 11th | Male | 40 | Handlers-cleaners | <=50K |
| **4** | 28 | 0 | 0 | 1 | Bachelors | Female | 40 | Prof-specialty | <=50K |

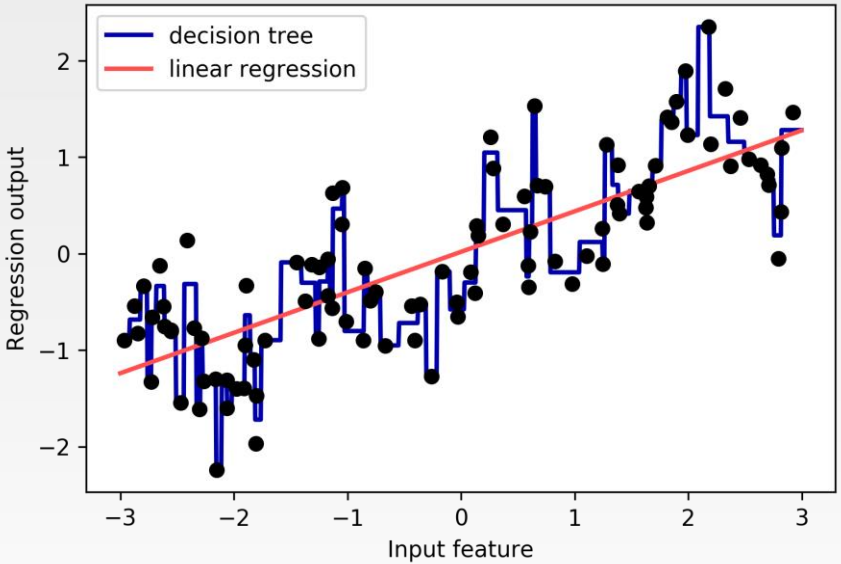
# One-hot-encoding at work

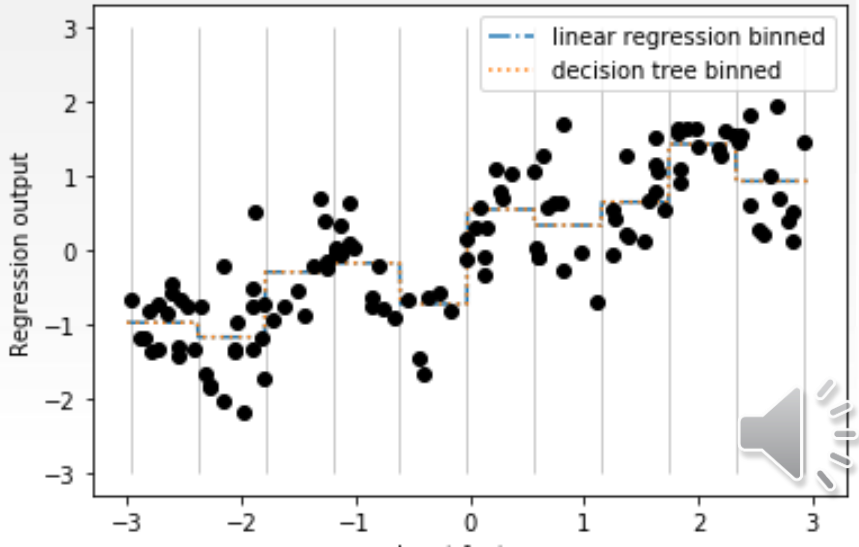
- sklearn: `get_dummies`
- Common mistakes:
  - Calling `get_dummies` separately on test data and training data when not all feature values are present in one of the sets
  - Not recognising that some numbers are actually encoding categorical features
  - Not checking the data for typos or different names used for the same category
  - Not removing the class labelling from the data before transformation

# Binning

- A column of continuous numbers has too many unique values to model effectively, so you automatically or manually assign the values to groups, to create a smaller set of discrete ranges.
- Replace a column of numbers with categorical values that represent specific ranges.
- A dataset has a few extreme values, all well outside the expected range, and these values have an outsized influence on the trained model.
- Binning has no effect for tree-based models
- Some clustering methods can sometimes help you decide on the number of bins

# Example

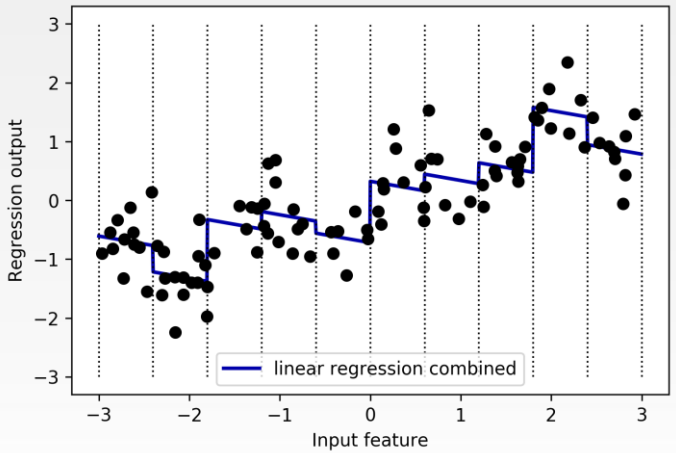

decision tree
linear regression
Regression output
Input feature
−3
−2
−1
0
1
2
3


linear regression binned
decision tree binned
Regression output
−3
−2
−1
0
1
2
3

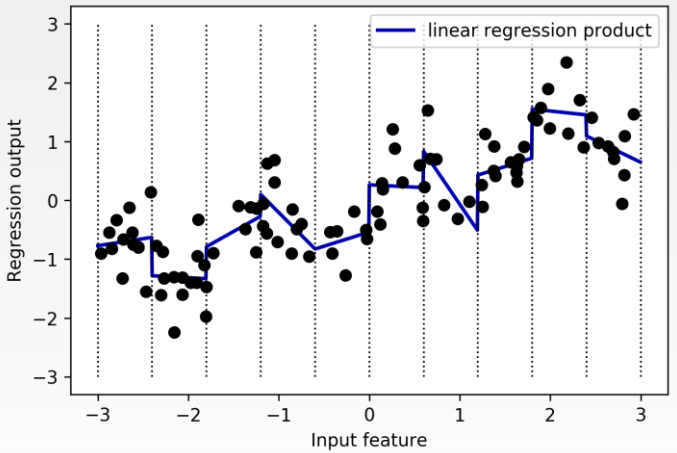
# Learning slopes

One way to increase the power of a linear model on binned data is to add the original feature back in. In this example this leads to an 11-dimensional data set.

To learn a different slope for each bin we can add an interaction or product feature that indicates which bin a data point is in and where it lies on the x-axis. This feature is a product of the bin indicator and the original feature




```
X_combined = np.hstack([X, X_binned])
```

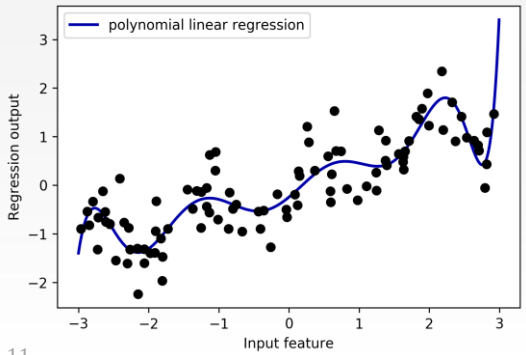


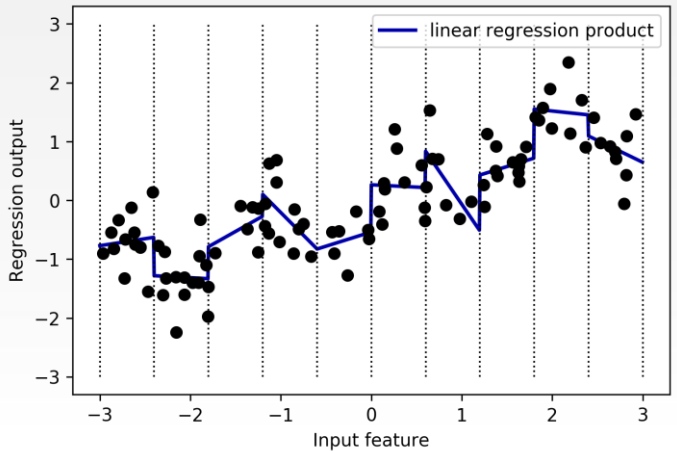

```
X_combined = np.hstack([X, X*X_binned])
```

# Polynomial features

- Given a feature x, we might want to consider $x^2$, $x^3$, $x^4$, and so on. This is implemented in `PolynomialFeatures` in the preprocessing module (of scikit-learn)
- Polynomial feature + linear regression = polynomial regression

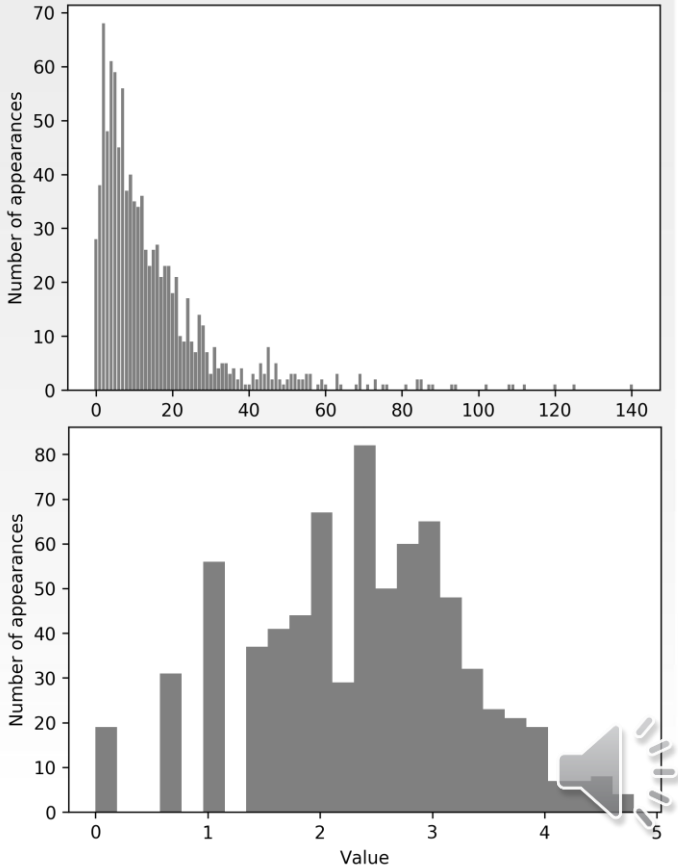
```
from sklearn.preprocessing import PolynomialFeatures

# include polynomials up to x ** 10:
# the default "include_bias=True" adds a feature that's constantly 1
poly = PolynomialFeatures(degree=10, include_bias=False)
poly.fit(X)
X_poly = poly.transform(X)
```



# Univariate nonlinear transformations

- You can transform features adding als sorts of operations on them: log, exp, or sin
- Linear models and neural networks (unlike tree-based models who are only affected by feature order) are very tied to the scale and distribution of each feature
- If there is a nonlinear relation between the feature and the target, that becomes hard to model (learn), particularly in regression
- The functions exp and log can help adjust the relative scales in the data so that they can be captured better by a linear model or neural network
- Why? Most models work best when each feature (and in regression also the target) is loosely Gaussian distributed
- The sin and cos functions can help adjust data that encodes periodic patterns



# Automatic feature selection

- Instead of adding features, in high-dimensional data sets it can be smart to remove those that are not useful
- Methods to know how good a feature is:
  - univariate statistics
  - model-based selection
  - iterative selection
  - Data should be split and methods applied on training data only. Keep test dta as is!
- Use expert knowledge

# Univariate statistics

- We compute whether there is a statistically significant relationship between each feature and the target
- The features that are related with the highest confidence are selected
- In case of classification this process is also known as **analysis of variance** (ANOVA)
- The method is **univariate** because it considers each feature individually
- => if a feature is informative only if combined with another feature it will be discarded

# Evaluation metrics and scoring

- Classification performance has been evaluated using accuracy
- Regression performance has been evaluated using $R^2$
  - in most cases using the default $R^2$ score is sufficient
- Business impact - the consequences of choosing a particular algorithm for a machine learning application
- Metrics for binary classification
- Metrics for multi class classification

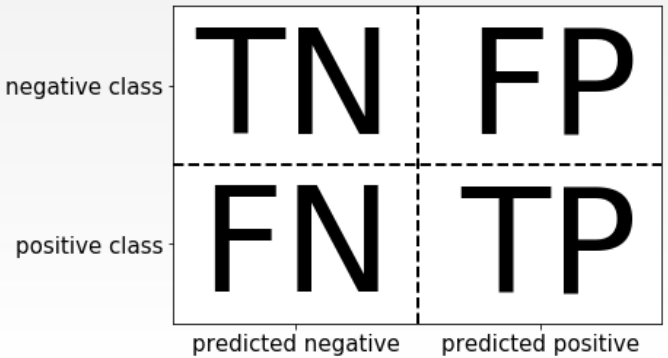


# Metrics for binary classification

The smear test scandal came to light after Limerick mother-of-two Vicky Phelan settled a High Court case for €2.5 million; the action was taken against the US laboratory which analysed her test. Phelan's 2011 smear test results were reviewed in 2014, where an error was found.

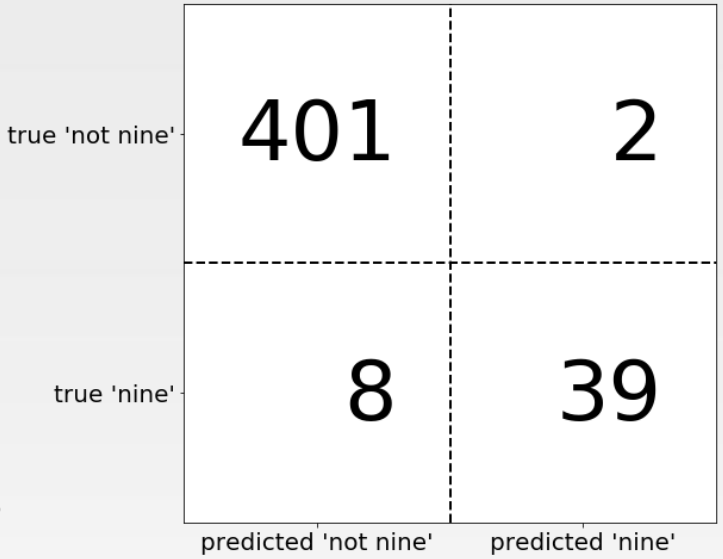
She wasn't told about the review or the error until 2017, three years after receiving a cervical cancer diagnosis.

- Positive class and negative class
- When is accuracy not a good measure of predictive performance?



# Confusion matrix

- Imbalanced datasets - datasets in which one class is much more frequent than the other
- Accuracy without knowing which class is the accuracy on is not meaningful in imbalanced datasets
- How good is the prediction when compared to frequency of class?
- Confusion matrix - a comprehensive way of representing the result when evaluating binary classification




$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

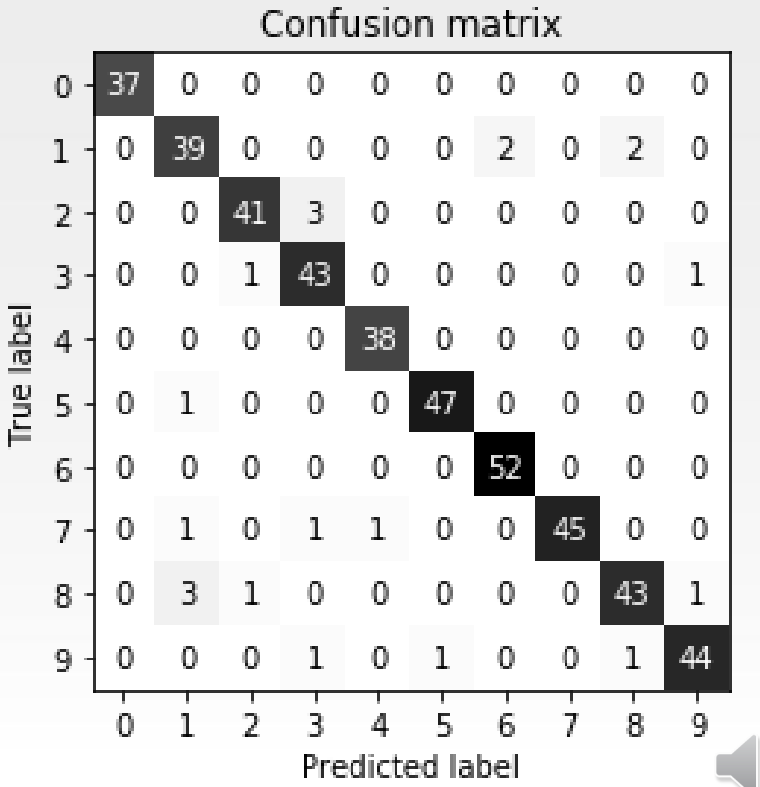
$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{Recall} = \frac{TP}{TP+FN}$$

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

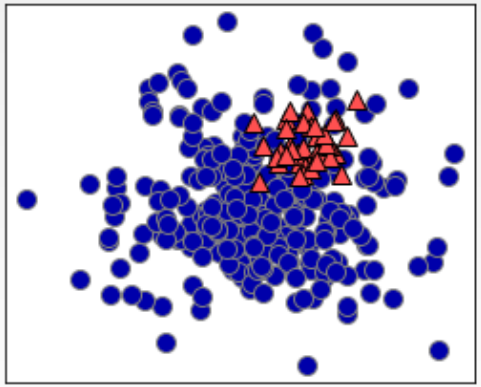
# Metrics for multiclass classification

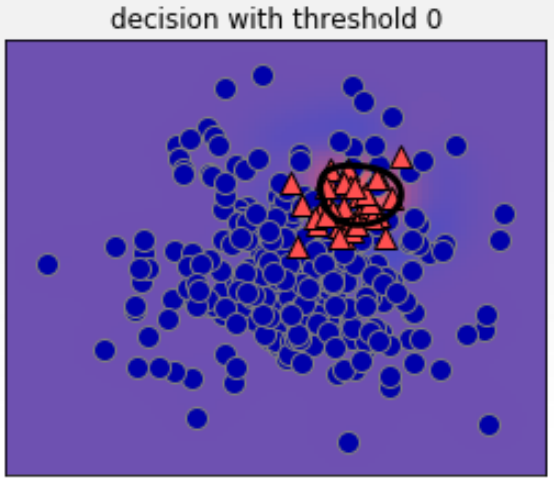
- All metrics for multi class classification are derived from binary classification metrics, but averaged over all classes
- **f-score** is computed once per class, with the class designated as the positive class, and all other classes as the negative class. Then the per-class f-scores are averaged using one of the following strategies: macro, weighted, micro



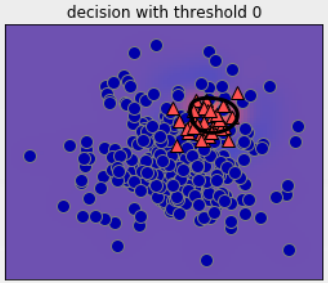
# Considering uncertainty

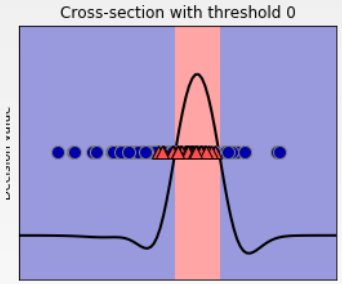
- Most classifiers provide a `decision_function` or a `predict_proba` method to assess degrees of certainty about predictions





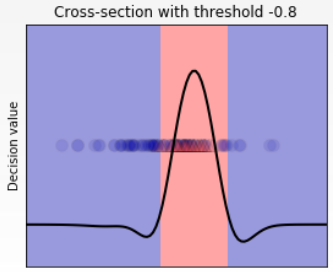
# Considering uncertainty




```
print(classification_report(y_test, svc.predict(X_test)))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.97 | 0.89 | 0.93 | 104 |
| 1 | 0.35 | 0.67 | 0.46 | 9 |
| avg / total | 0.92 | 0.88 | 0.89 | 113 |

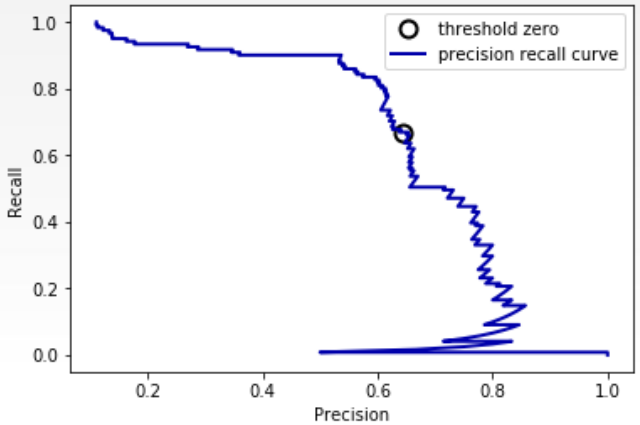



```
y_pred_lower_threshold = svc.decision_function(X_test) > -.8

print(classification_report(y_test, y_pred_lower_threshold))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 0.82 | 0.90 | 104 |
| 1 | 0.32 | 1.00 | 0.49 | 9 |
| avg / total | 0.95 | 0.83 | 0.87 | 113 |

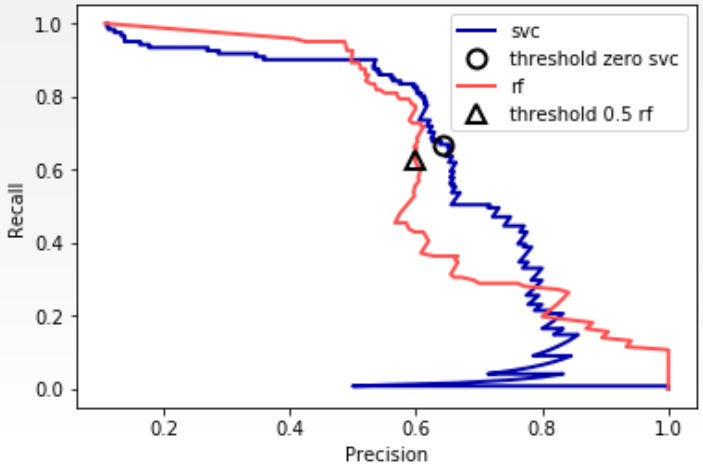
# Precision-recall curves

- Setting a requirement on a classifier like 90% recall is often called setting the **operating point**
- Not always clear what the operating point should be, therefore can be useful to see all possible trade-offs of precision and recalls at once -> **precision-recall curve**
- can be done in sklearn.metrics and it needs the ground truth labelling of a test set and predicted uncertainties from the learned model – see text book



# Precision-recall curves for predict_proba

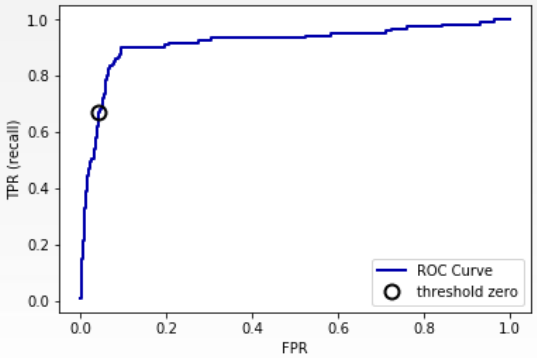
```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, random_state=0, max_features=2)
rf.fit(X_train, y_train)

# RandomForestClassifier has predict_proba, but not decision_function
precision_rf, recall_rf, thresholds_rf = precision_recall_curve(y_test,
rf.predict_proba(X_test)[:, 1])
```

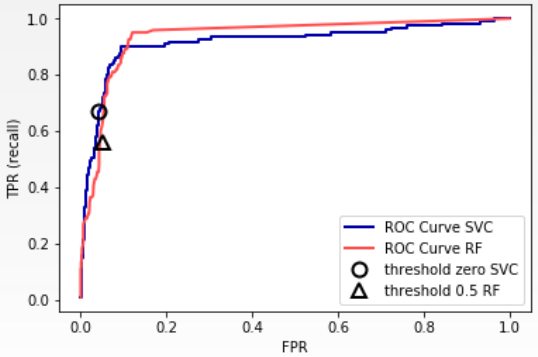


# Receiver operating characteristics (ROC)

- **ROC curve** considers all possible thresholds for a given classifier but it shows the false positive rate against the true positive rate FPR= FP/(FP+TN)
- **AUC** - area under the curve **roc_auc_curve** function

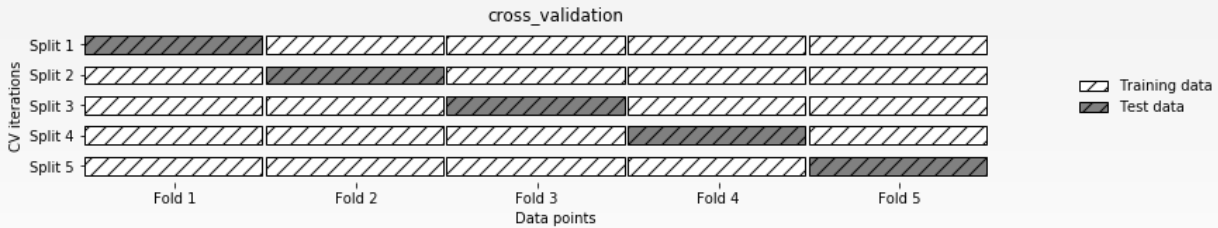
```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, svc.decision_function(X_test))
```



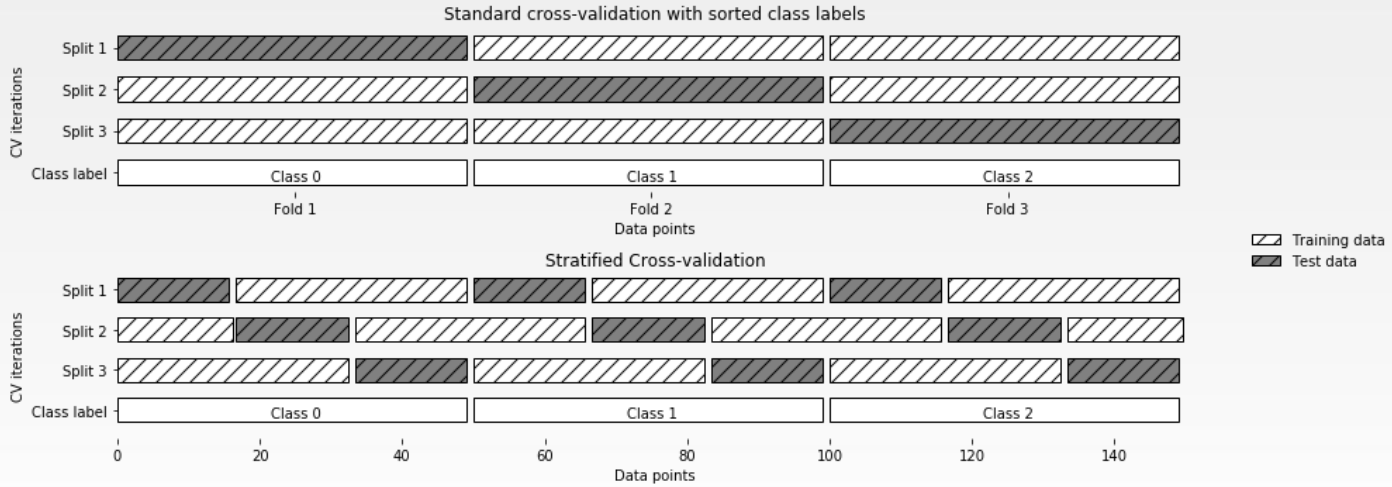
# Cross-validation

- Statistical method for evaluating generalisation performance.
- K-fold cross-validation



# Stratified k-Fold validation

- Split the data so that the proportion of classes is maintained



# More control over cross-validation

- The cv parameter allows you to choose the number of folds
  - cross_validate(..., cv = 10, ...)
- Instead of number of folds we can as a cv parameter choose a fold object, which allows for more control
  - KFold, StratifiedKFold

- Leave one-out cross-validation
  - For each split, you pick a single data point to be the test set
  - Sometimes provides better accuracy estimates on small data sets
  - cv = LeaveOneOut()

# Pro- and con- to cross-validation

- Compared to train_test_split, the measured accuracy depends less on randomness
- Having multiple splits of the data provides some information about how sensitive the model is to the selection of the training dataset. So we may learn something about the variance from using the particular model on this data set.
- It provides a more effective use of data. We may use all data for training
- High computational cost
- Cross-validation is not a way to build a model that can be applied to new data - only to see how good can a model be



# Grid search – finding the best hyperparameters

- How to improve the accuracy of a model?
- How to adjust the parameters in the learning models?
- Grid search is a way to try possible combinations of parameters
- Example: kernel SVM with an RBF kernel. There are two important parameters: the kernel `gamma` and the regularisation parameter, C. We want to try values 0.001, 0.01, 0.1, 1, 10 for both parameters

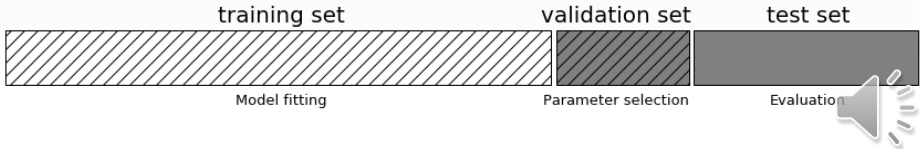


# Simple grid search

- We can implement a simple grid search by using for loops over the values of the parameters and then training and evaluating a classifier for each combination

```
# naive grid search implementation
from sklearn.svm import SVC
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, random_state=0)
print("Size of training set: {}   size of test set: {}".format(X_train.shape[0], X_test.shape[0]))
best_score = 0
for gamma in [0.001, 0.01, 0.1, 1, 10, 100]:
    for C in [0.001, 0.01, 0.1, 1, 10, 100]:
        # for each combination of parameters, train an SVC
        svm = SVC(gamma=gamma, C=C)
        svm.fit(X_train, y_train)
        # evaluate the SVC on the test set
        score = svm.score(X_test, y_test)
        # if we got a better score, store the score and parameters
        if score > best_score:
            best_score = score
            best_parameters = {'C': C, 'gamma': gamma}
print("Best score: {:.2f}".format(best_score))
print("Best parameters: {}".format(best_parameters))
```

```
Size of training set: 112   size of test set: 38
Best score: 0.97
Best parameters: {'C': 100, 'gamma': 0.001}
```



# The non-naive way to grid search

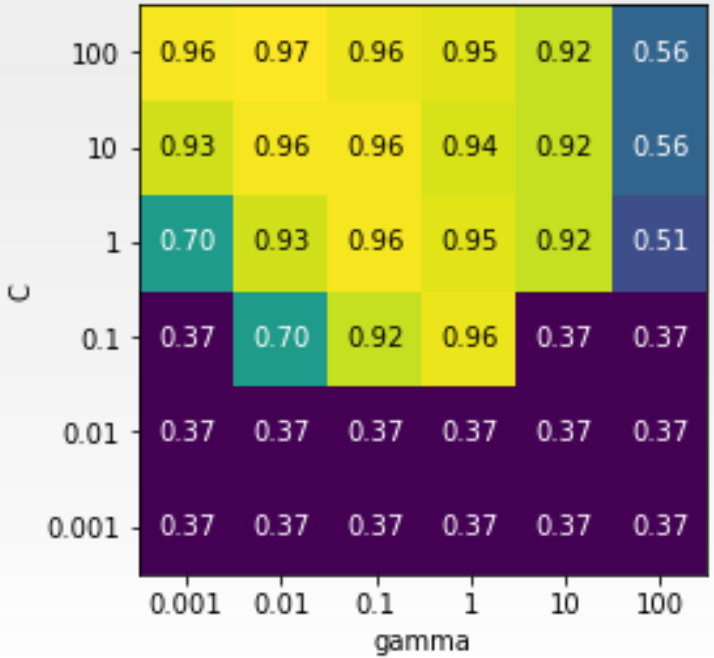
- `GridSearchCV`
  - `param_grid` is a list of dictionaries
- Each dictionary represent a grid for a particular parameter (in SVC the chosen kernel)

```
param_grid = [{'kernel': ['rbf'],
               'C': [0.001, 0.01, 0.1, 1, 10, 100],
               'gamma': [0.001, 0.01, 0.1, 1, 10, 100]},
              {'kernel': ['linear'],
               'C': [0.001, 0.01, 0.1, 1, 10, 100]}]


grid_search = GridSearchCV(SVC(), param_grid, cv=5,
                           return_train_score=True)
grid_search.fit(X_train, y_train)
print("Best parameters: {}".format(grid_search.best_params_))
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))

Best parameters: {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
Best cross-validation score: 0.97
```



# Heat maps of two parameter grids

- Start with a relatively coarse and small grid
- Inspect the results and then expand




uib.no