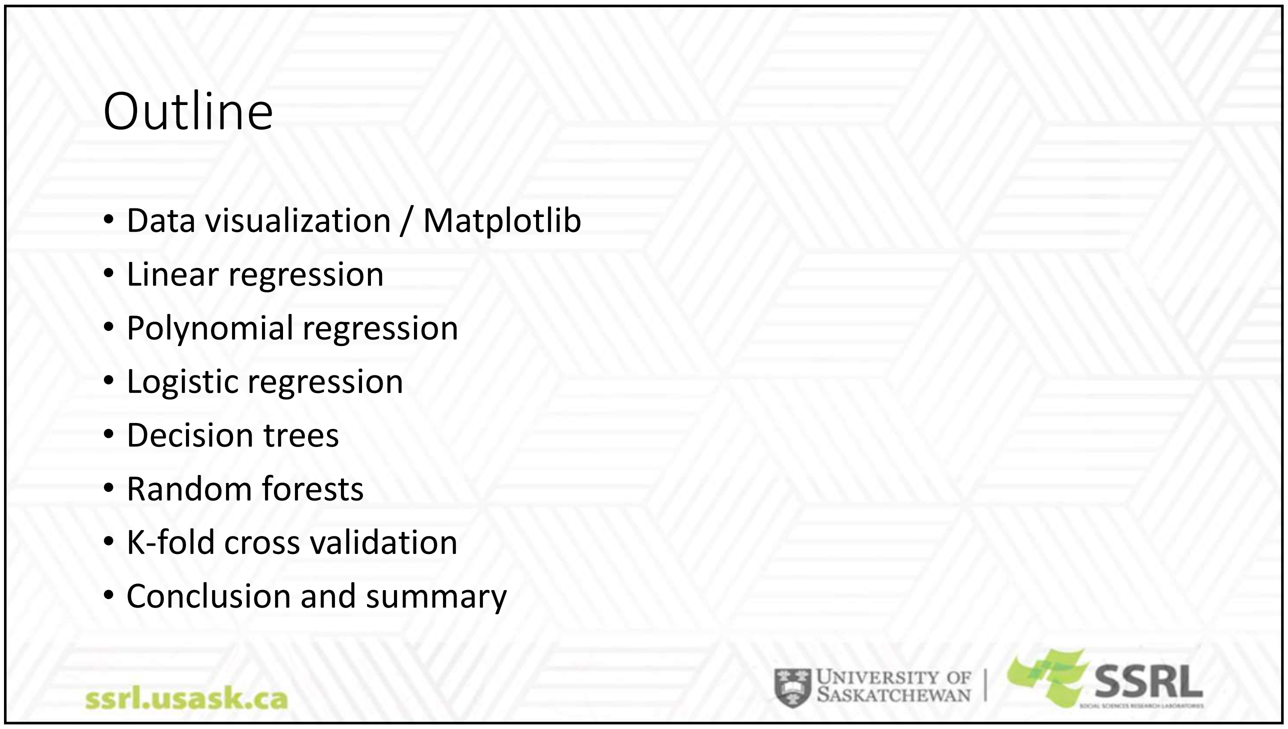
Introduction to Applied Machine Leaning II

David Pinelle, Ph.D., Computational Research Manager
Social Sciences Research Laboratories
University of Saskatchewan
Saskatoon, Saskatchewan, Canada

ssrl.usask.ca



Outline

- Data visualization / Matplotlib
- Linear regression
- Polynomial regression
- Logistic regression
- Decision trees
- Random forests
- K-fold cross validation
- Conclusion and summary

ssrl.usask.ca

Code samples and assignment

- Available on Github:
- https://github.com/pinelle

Machine learning steps

- Gathering data
- Preparing data
- Model building
- Parameter tuning
- Evaluation
- Prediction

Machine learning steps

- Gathering data
- Preparing data
- **Model building**
- **Parameter tuning**
- **Evaluation**
- **Prediction**

ssrl.usask.ca

No free lunch theorem

- No one algorithm works best for every problem
- It is especially relevant for supervised learning
- For example, cannot say that random forest is better than gradient boosted regression trees
- There are too many factors at play, such as the size and structure of the dataset and the current problem
- Pick algorithms that are appropriate to the dataset and problem / task, and test and evaluate to find the best one

ssrl.usask.ca

Technology for this workshop

- Python 3 version of Anaconda
- Spyder as the IDE

- Spyder is bundled with Anaconda
- Anaconda is available at https://www.anaconda.com/distribution/

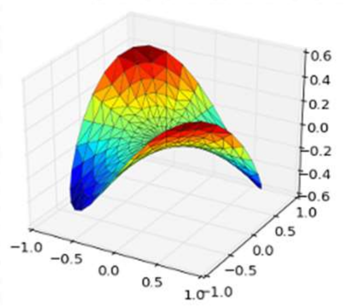
ANACONDA

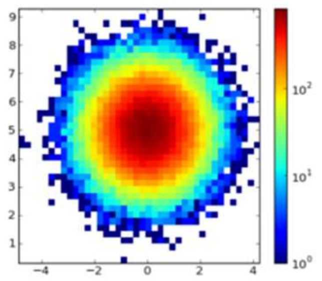
SPYDER

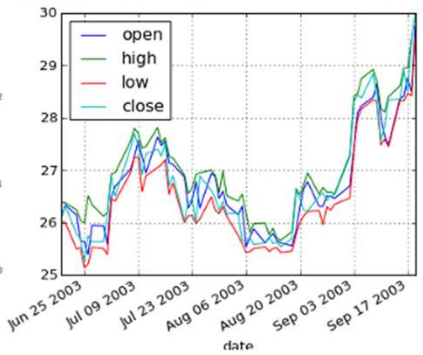
ssrl.usask.ca

UNIVERSITY OF SASKATCHEWAN | SSRL SOCIAL SCIENCES RESEARCH LABORATORIES

Data visualization





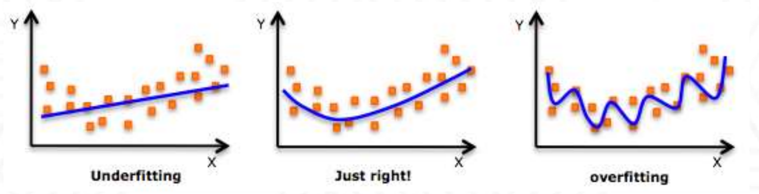


ssrl.usask.ca

UNIVERSITY OF SASKATCHEWAN | SSRL SOCIAL SCIENCES RESEARCH LABORATORIES

Data visualization

- Needed to understand the model and how the data is distributed
- Also helps communicate results in a meaningful way



University of Saskatchewan | SSRL Social Sciences Research Laboratories

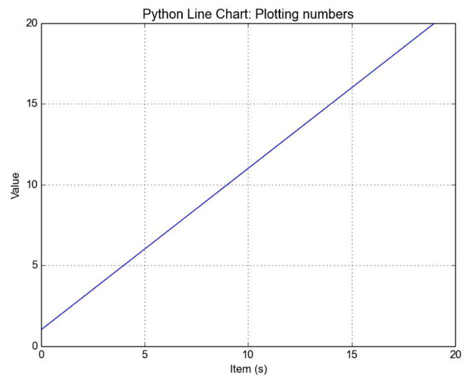
Matplotlib

- Very popular 2D visualization library
- Wide variety of plotting and charting options
- Easy to feed in results from other modules (like Pandas, scikit-learn, NumPy, SciPy, etc.)
- Built in functions translate data into charts, manipulate plots, add labels, etc.

University of Saskatchewan | SSRL Social Sciences Research Laboratories

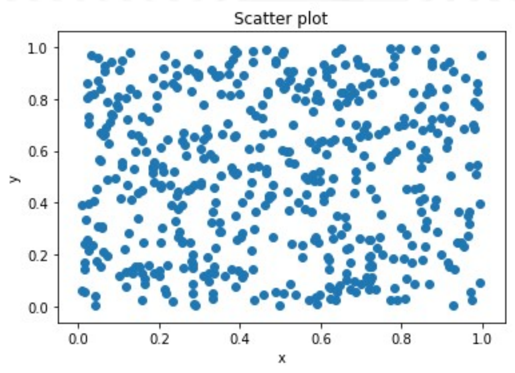
Matplotlib: Line Chart

```
import numpy as np
import matplotlib.pyplot as plt

x = np.arange(0.0, 20.0, 1)
y = np.arange(20)
plt.plot(x, y)

plt.xlabel('Item (s)')
plt.ylabel('Value')
plt.title('Python Line Chart: Plotting numbers')
plt.grid(True)
plt.show()
```



Matplotlib: Scatterplot

```
import numpy as np
import matplotlib.pyplot as plt

# Create data
N = 500
x = np.random.rand(N) #create arrays of shape 500
y = np.random.rand(N) #randomly populate: 0 to 1

# Plot
plt.scatter(x, y)
plt.title('Scatter plot')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



Matplotlib.pyplot

- Labels

```
plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
```

- Add grid

```
plt.grid(True)
```

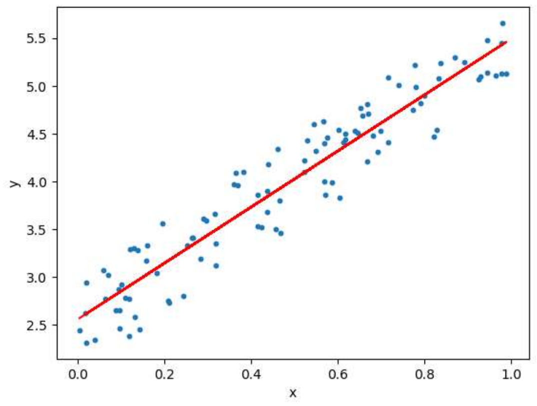
- Create chart

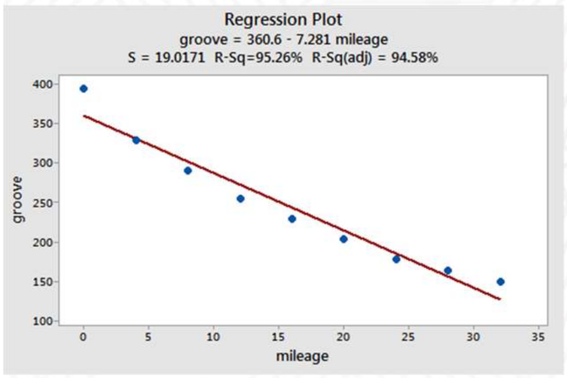
```
plt.bar(....)
plt.scatter(...)
plt.plot(...)
```

- Display chart

```
plt.show()
```

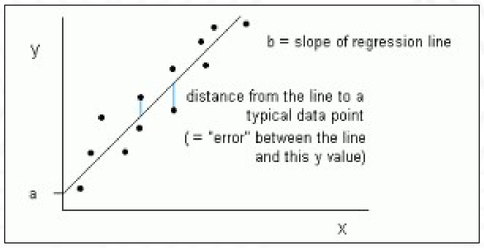
Linear regression



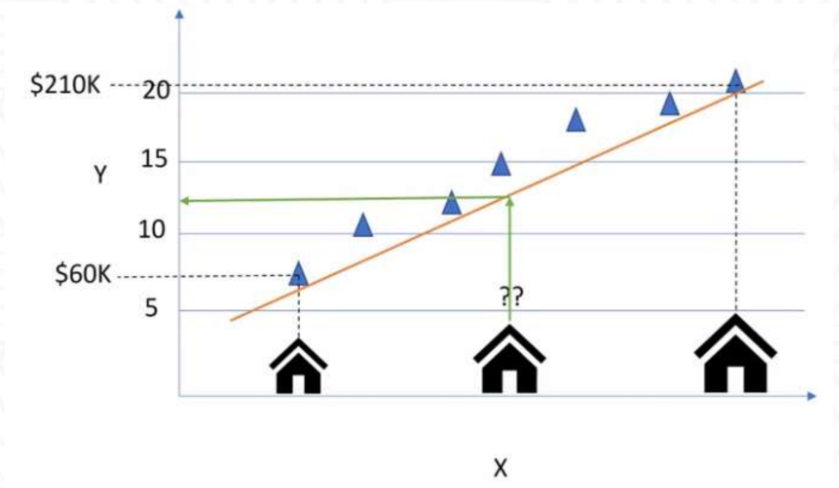


Linear regression algorithms

- Linear Regression is used in problems where the output (y variable) is continuous (i.e. can take on any value), e.g sales, stocks trading, etc.
- It uses an ordinary least squares method fitting the best line that minimizes the sum of squared errors between the predicted and actual data points.


y
b = slope of regression line
distance from the line to a typical data point
(= "error" between the line and this y value)
a
x

Linear regression



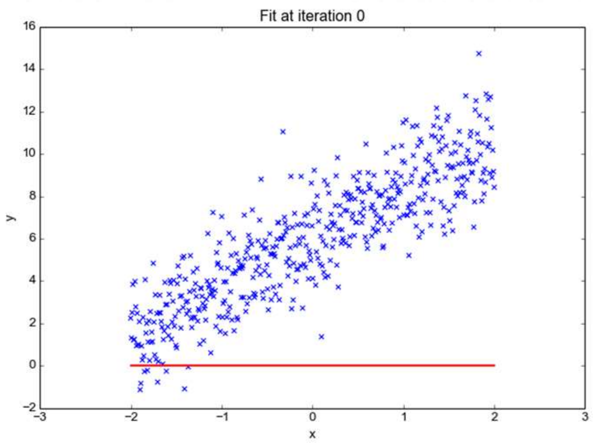
Simple Linear Regression (OLS)

- **Y = mx + b**
- Where b is the intercept (y value when x=0) and m is the slope of the line
- The y and x variables always remain the same, since they are plotted to fit the data and cannot be changed.
- The linear regression algorithm chooses the most optimal value for the intercept and the slope (in two dimensions)
- It fits multiple lines to the plot, and chooses the one that minimizes error the most
- The linear regression algorithm fits and evaluates multiple lines against the data points
  - It returns the corresponding slope and interceptIt selects the line that minimizes the sum of squared errors

ssrl.usask.ca

University of Saskatchewan | SSRL Social Sciences Research Laboratories

Simple Linear Regression (OLS)



ssrl.usask.ca

University of Saskatchewan | SSRL Social Sciences Research Laboratories

R-squared (R^2)

- R-squared is a statistical measure of how close the data are to the fitted regression line
- It is the percentage of the response variable variation that is explained by a linear model. Or: **R-squared = Explained variation / Total variation**
- R-squared is always between 0 and 100%
- 0% indicates that the model explains none of the variability of the response data around its mean
- 100% indicates that the model explains all the variability of the response data around its mean
- The higher the R-squared, the better the model fits your data

Simple linear regression

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes = datasets.load_diabetes()

# Use all rows, only one feature
diabetes_X = diabetes.data[:, np.newaxis, 2]
```

Example by Jaques Grobler

Simple linear regression

```
# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]
```

Example by Jaques Grobler

Simple linear regression

```
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)

# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# R-squared score: 1 is perfect prediction
print(‘R-squared: %.2f' % r2_score(diabetes_y_test, diabetes_y_pred))
R-squared: 0.47
```

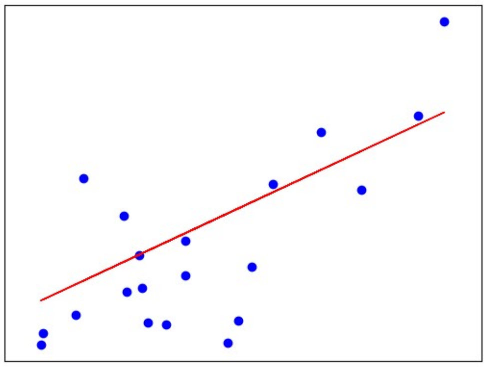
Example by Jaques Grobler

Simple linear regression

```
# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test,  color='blue')
plt.plot(diabetes_X_test, diabetes_y_pred, color='red',
linewidth=1)

plt.xticks(())
plt.yticks(())

plt.show()
```

Example by Jaques Grobler

Simple linear regression



Simple linear regression

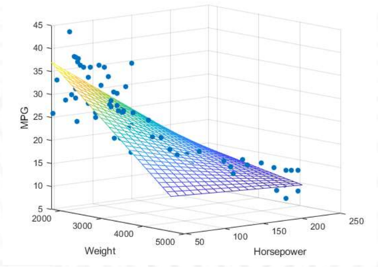
- Advantages
  - Very simple and easy to use
  - Easy to explain and easy for others to understand
  - Works well when there is a linear relationship between factor(s) and outcome
- Disadvantages
  - Linear relationships often do not exist in many real-world datasets
  - Very sensitive to outliers
  - Tends to underfit data

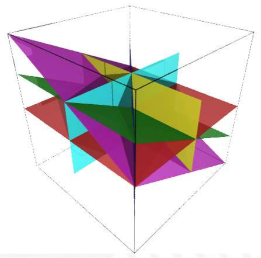
Multiple linear regression

- Use when there is more than one feature (x variable, aka independent variable).
- There should be no multicollinearity – features (independent variables) should not be highly correlated
- Should be linear relationship between each factor (x) and the dependent variable (y)

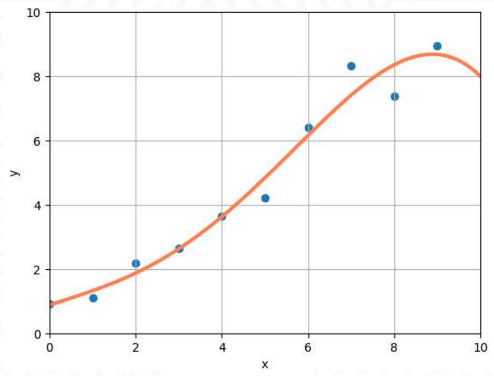
Multiple linear regression

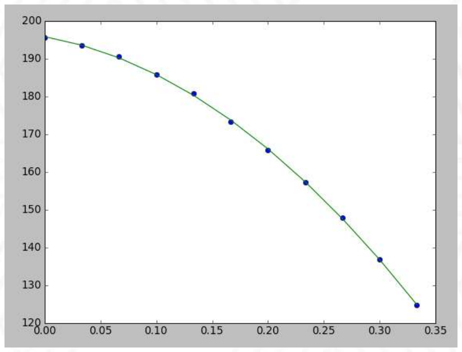
- Charting is needed to confirm assumptions
- Scikit-learn coding is very similar, regardless of the number of factors
- Conceptually, the plot can be a plane (1 independent variable), 3-d plane (2 variables), or a hyperplane (greater than 2)





Polynomial regression





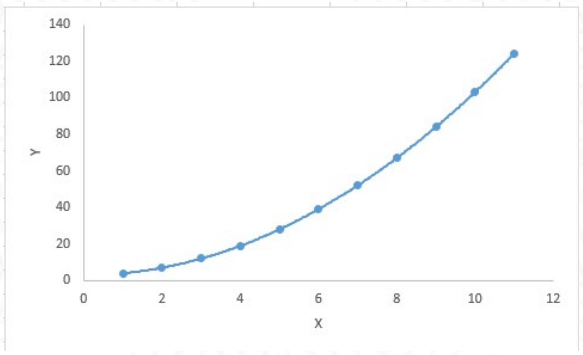
Polynomial regression

- A regression equation is a polynomial regression equation if the power of independent variable is more than 1. The equation below represents the polynomial equation:
- $y = a + b_1x + b_2x^2 + ... + b_nx^n$ (polynomial order n)
- In this regression technique, the best fit line is not a straight line. It is a curve that fits the data points.

ssrl.usask.ca

University of Saskatchewan | SSRL

Polynomial regression

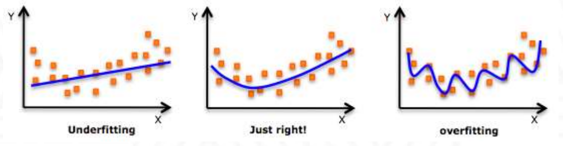


ssrl.usask.ca

University of Saskatchewan | SSRL

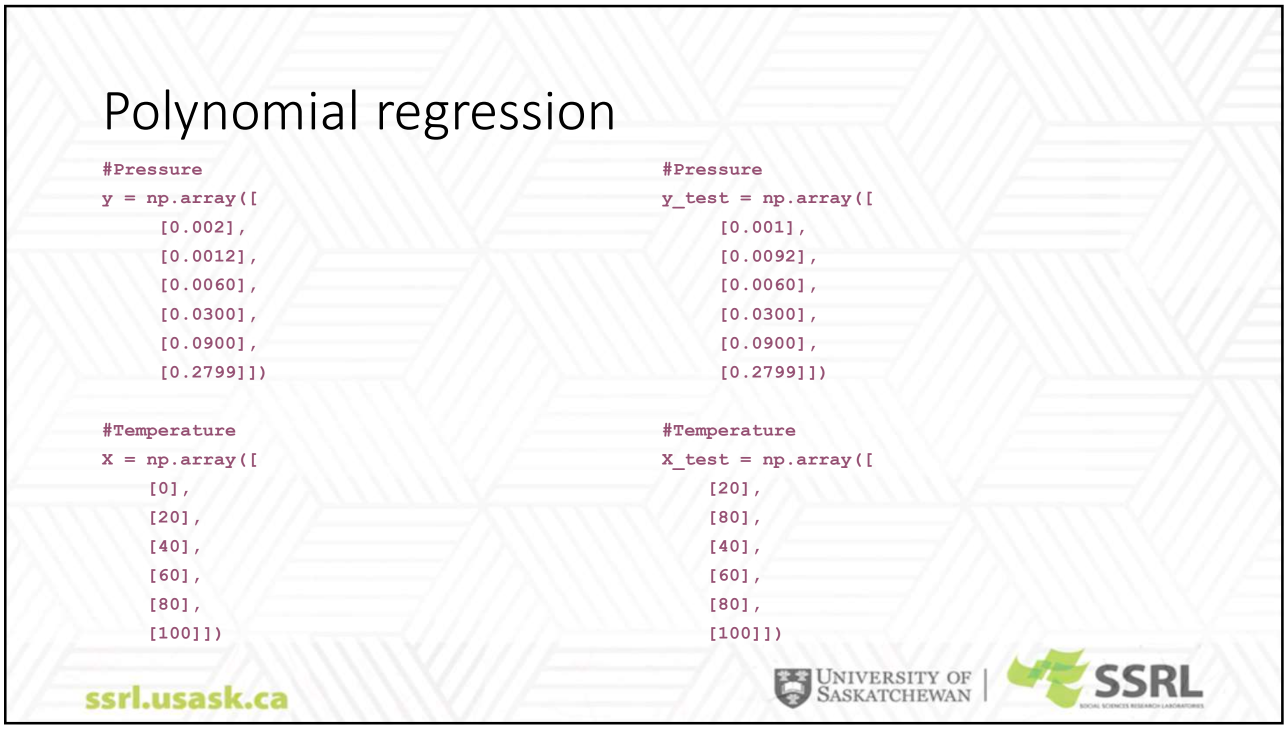
Polynomial regression

- While there might be a temptation to fit a higher degree polynomial to get a lower error, this can result in overfitting.
- Always plot the relationships to see the fit and to make sure that the curve fits the data properly.



Polynomial regression

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

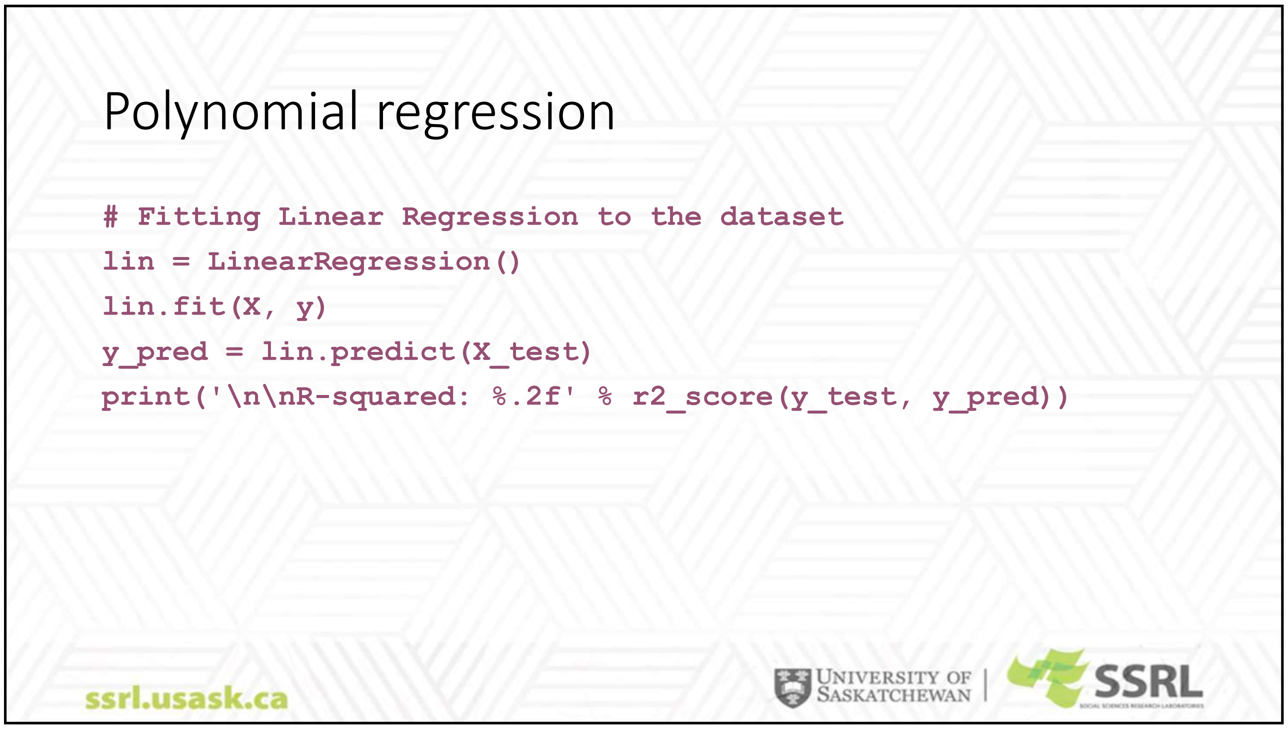


Polynomial regression

```
#Pressure
y = np.array([
     [0.002],
     [0.0012],
     [0.0060],
     [0.0300],
     [0.0900],
     [0.2799]])

#Temperature
X = np.array([
    [0],
    [20],
    [40],
    [60],
    [80],
    [100]])
```

```
#Pressure
y_test = np.array([
     [0.001],
     [0.0092],
     [0.0060],
     [0.0300],
     [0.0900],
     [0.2799]])

#Temperature
X_test = np.array([
    [20],
    [80],
    [40],
    [60],
    [80],
    [100]])
```

ssrl.usask.ca



Polynomial regression

```
# Fitting Linear Regression to the dataset
lin = LinearRegression()
lin.fit(X, y)
y_pred = lin.predict(X_test)
print('\n\nR-squared: %.2f' % r2_score(y_test, y_pred))
```

ssrl.usask.ca

Polynomial regression

```
# Fitting Polynomial Regression to the dataset
poly = PolynomialFeatures(degree = 4)
X_poly = poly.fit_transform(X)
poly.fit(X_poly, y)
lin2 = LinearRegression()
lin2.fit(X_poly, y)
```

Polynomial regression

```
# Visualising the Linear Regression results
plt.scatter(X, y, color = 'blue')

plt.plot(X, lin.predict(X), color = 'red')
plt.title('Linear Regression')
plt.xlabel('Temperature')
plt.ylabel('Pressure')

plt.show()
```

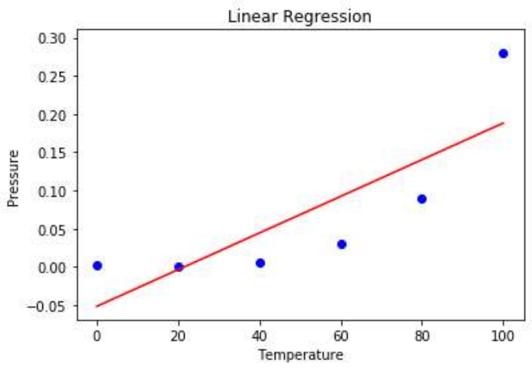
Polynomial regression

```
# Visualising the Polynomial Regression results
y_pred = lin2.predict(poly.fit_transform(X_test))
print('R-squared: %.2f' % r2_score(y_test, y_pred))

plt.scatter(X, y, color = 'blue')
plt.plot(X, lin2.predict(poly.fit_transform(X)), color = 'red')
#Plot titles
plt.show()
```

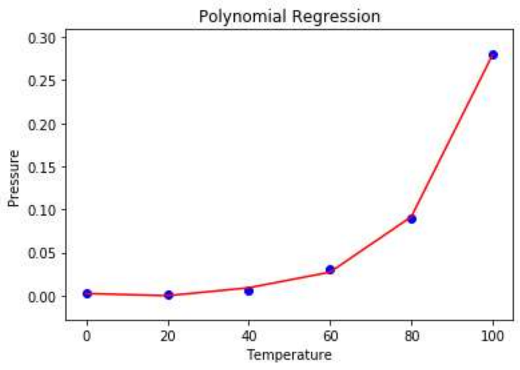
ssrl.usask.ca

Polynomial regression

- R-squared: 0.43



- R-squared: 0.88

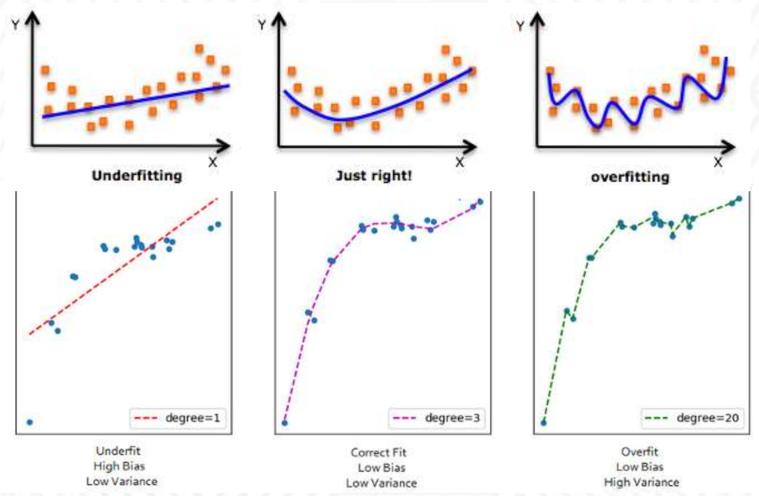


ssrl.usask.ca

Polynomial regression

- Advantages
  - Significantly more flexible than simple linear regression
  - Can model complex relationships and fits a wide range of curvatures
  - Easy to explain and interpret
- Disadvantages
  - Too sensitive to outliers
  - Can overfit data
  - May be outperformed by more complex algorithms

Underfitting and overfitting



Assignment

- Copy and paste the assignment into a new Anaconda window
- The code is available at: **https://github.com/pinelle**

- The code runs polynomial regression on a randomly selected dataset
- The data is plotted on a scatterplot, and the regression line is shown
- Continue to run the code approximately 20 times, and note how the data distribution affects the R-squared value

Assignment

```
#create data
X, y = make_regression(n_samples = 300, n_features=1, noise=8,
bias=2)
y2 = y**2
```

Assignment

```
#create model
poly_features = PolynomialFeatures(degree =5)
X_poly = poly_features.fit_transform(X)
poly_model = LinearRegression()
poly_model.fit(X_poly, y2)
y_pred = poly_model.predict(X_poly)
```

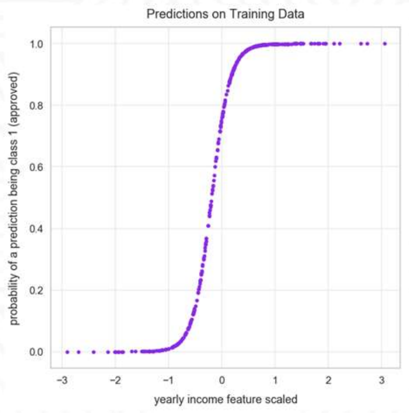
ssrl.usask.ca

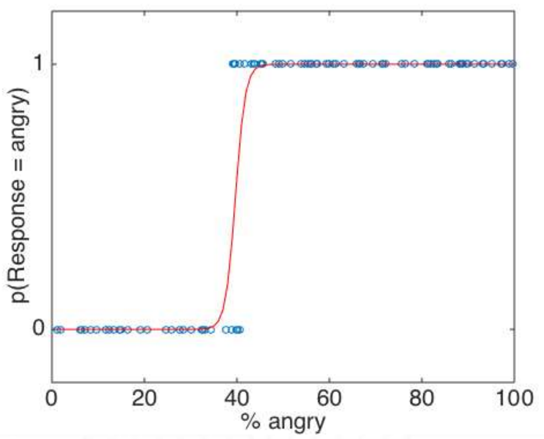
Assignment

- Continue to run the code approximately 20 times, and note how the data distribution affects the R-squared value
- How was the data distributed (i.e. in the scatterplot) when:
  - The R-squared value was high > 90%
  - The R-squared value was low < 60%
- What caused these differences?

ssrl.usask.ca

Logistic regression





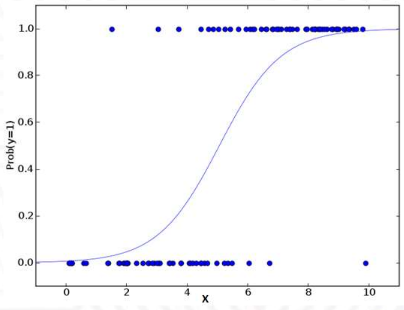
Logistic regression

- Logistic Regression is a Machine Learning classification algorithm that uses probability to classify a categorical dependent variable.
- The dependent variable is a binary variable that contains data coded as 1 (yes, success) or 0 (no, failure).
  - Spam / not spam, cancer / not cancer, diabetes / not diabetes, purchase item? yes / no, etc.
- The independent variables should be independent of each other. In other words, they should have little or no multicollinearity.

Logistic regression

- The logistic regression model predicts P(Y=1) as a function of X.
- It predicts the probability of occurrence of an event by fitting data to a logistic function, which has an ‘S’ shaped curve.

$$P(1) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 * x_1 + \theta_2 * x_2)}}$$



Logistic regression and probability thresholds

- Logistic regression outputs probabilities
- If the probability ‘p’ is greater that 0.5
  - The data is labeled ‘1’
- If the probability ‘p’ is less than 0.5
  - The data is labeled ‘0’

Logistic regression

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LogisticRegression

#x contains bid price, y contains win (1) / loss (0)
x = np.array([100,120,150,170,200,200,202,203,205,210,215,250,270,300,305,310])
y = np.array([1,1,1,1,1,1,1,0,1,0,0,0,0,0,0,0])

x_test = np.array([120,155,174, 200,202,203,215, 250 ,400, 510, 660, 529, 660,
710, 888, 900])
y_test = np.array([1,1,1,1,1,0,1,0,0,0,0,0,0,0,0,0])
```

Logistic regression

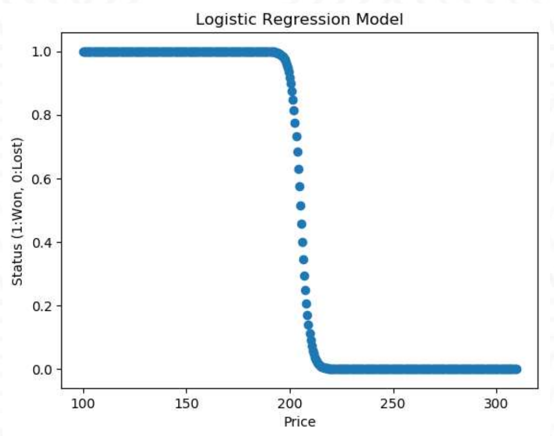
```
#Convert a 1D array to a 2D array in numpy
X = x.reshape(-1,1)
X_test = x_test.reshape(-1,1)

logreg = LogisticRegression(C=1.0, solver='lbfgs', multi_class='ovr')
logreg.fit(X, y)

score = logreg.score(X_test, y_test)
print(score)
0.875
```

Logistic regression

```
prices = np.arange(100, 310, 0.5) #create values, step .5, for plot
probabilities= []
for i in prices:
    p_loss, p_win = logreg.predict_proba([[i]])[0]
    probabilities.append(p_win)

plt.scatter(prices,probabilities) #display linear regression model
plt.title("Logistic Regression Model")
plt.xlabel('Price')
plt.ylabel('Status (1:Won, 0:Lost)')
```

Logistic regression

- Accuracy: .875



Multiclass logistic regression

- Logistic regression can handle multiple classes (i.e. labels)
- By default, it uses a *one versus rest* algorithm
  - It build a binary model for each class. For example, if there are 10 classes, 10 models are built.
  - When an unknown case is introduced
    - Each model evaluates whether the case belongs to the corresponding class or not
    - The algorithm assigns the case to the most likely outcome (with the highest probability)

ssrl.usask.ca

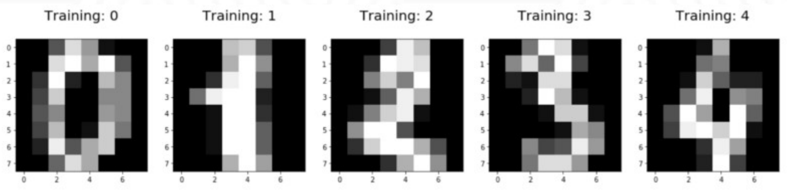
Multiclass logistic regression

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_digits

digits = load_digits()
print('Image Data Shape' , digits.data.shape) # 1797 images, 8 x 8 pixels = 64
Image Data Shape (1797, 64)
print("Label Data Shape", digits.target.shape) #1797 labels (integers from 0-9)
Label Data Shape (1797,)
```

ssrl.usask.ca

Multiclass logistic regression

```
#plot the first five images in the dataset
plt.figure(figsize=(20,4))
for index, (image, label) in enumerate(zip(digits.data[0:5], digits.target[0:5])):
        plt.subplot(1, 5, index + 1)
        plt.imshow(np.reshape(image, (8,8)), cmap=plt.cm.gray)
        plt.title('Training: %i\n' % label, fontsize = 20)
```



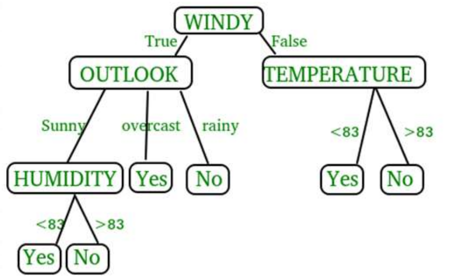
Multiclass logistic regression

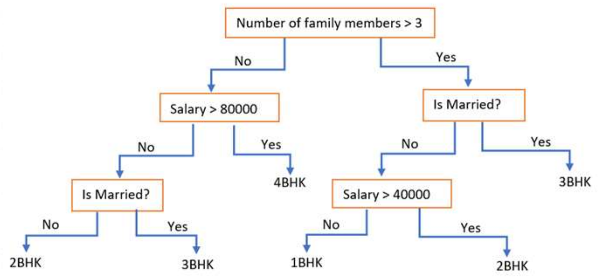
```
x_train, x_test, y_train, y_test = train_test_split(digits.data,
digits.target, test_size=0.25, random_state=0)

logisticRegr = LogisticRegression()
logisticRegr.fit(x_train, y_train)

score = logisticRegr.score(x_test, y_test)
print(score)
0.9533333333333334
```

Logistic regression

- Advantages
  - Easy to implement
  - Very efficient to train
  - Highly interpretable
- Disadvantages
  - Underperforms when there are nonlinear decision boundaries
  - Since the outcome is discrete, it can only predict a categorical outcome
  - Very vulnerable to overfitting
  - May be outperformed by more complex algorithms

Decision trees



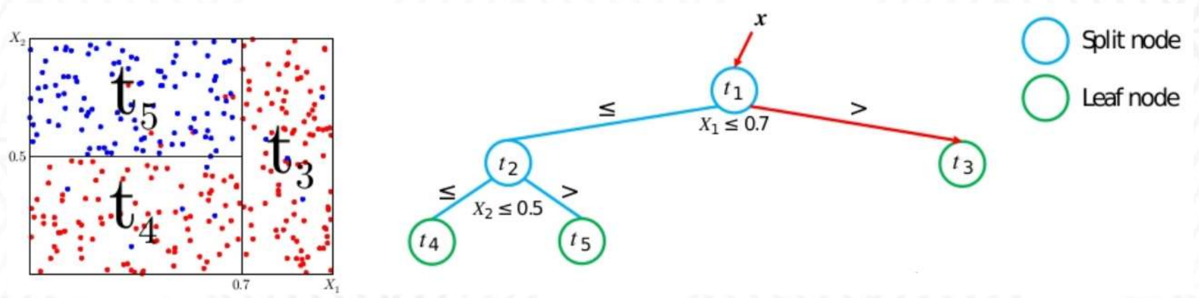


Decision trees

- The decision tree algorithm generates decision trees from the training data to solve classification and regression problems
- Decision tree data is recursively partitioned according to certain parameters
- New points are evaluated based on which region of the dataset they are plotted in
- The tree can be explained by two entities
  - The <u>decision nodes</u> are where the data is split. The top node represents the entire dataset, and is called the <u>root node</u>.
  - The <u>leaves</u> are the decisions or the final outcomes

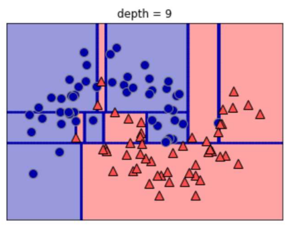
Decision trees

- Decision trees ask a sequence of if / else questions, leading to a decision
- Goal is to get to the correct answer as quickly as possible



Decision trees

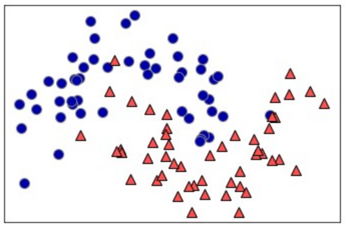
- Overfitting is one of the main challenges in using decision trees
- By default, a decision tree will be built so that each terminal leaf contains a single sample of data
- This overfits the training data, making sure every data point in the training set is 100% accurate
- …However, the model may not generalize well to other data sets

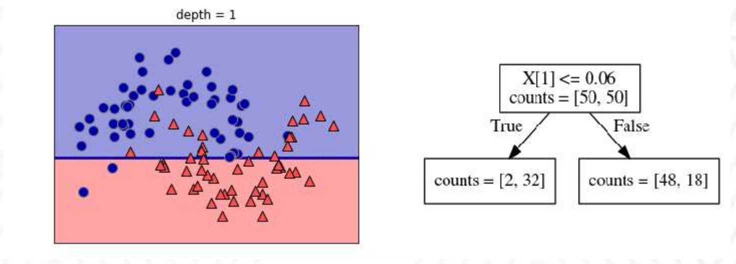


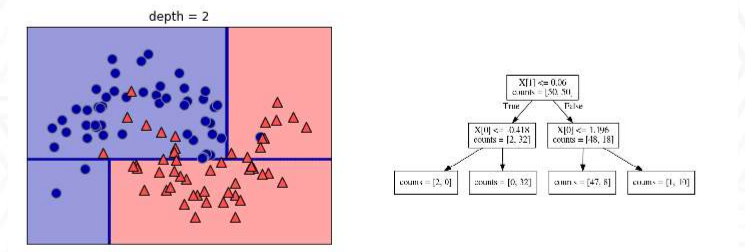
Decision trees

- There are two main strategies for dealing with overfitting
  - Post-pruning: Build the tree and remove nodes that contain little information. *This is not currently supported in scikit-learn.*
  - Pre-pruning: Limit the creation of the tree before it's built. Scikit-learn supports three main approaches
    - Limit the maximum depth of the tree
    - Limiting the maximum number of leaves (i.e. terminal nodes)
    - Requiring a minimum number of point in a node to keep splitting it

Decision trees



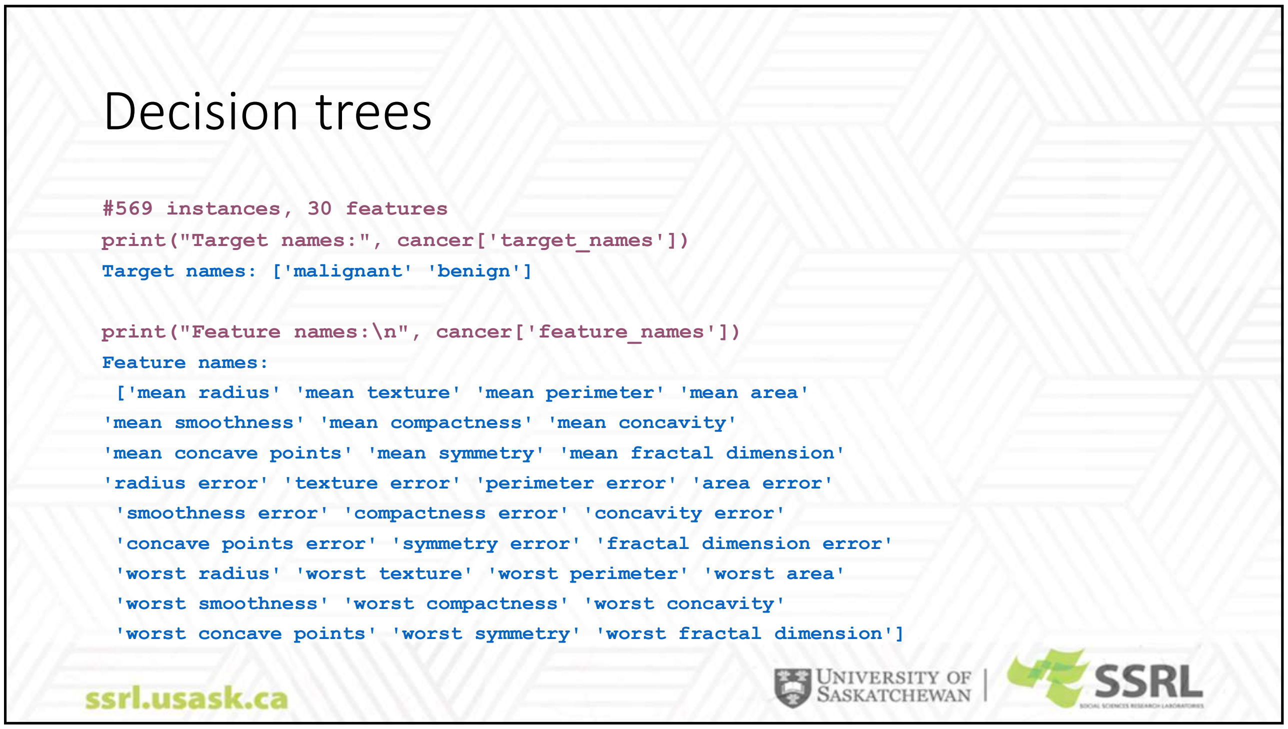




Decision trees

- Wisconsin breast cancer diagnostic dataset
- Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

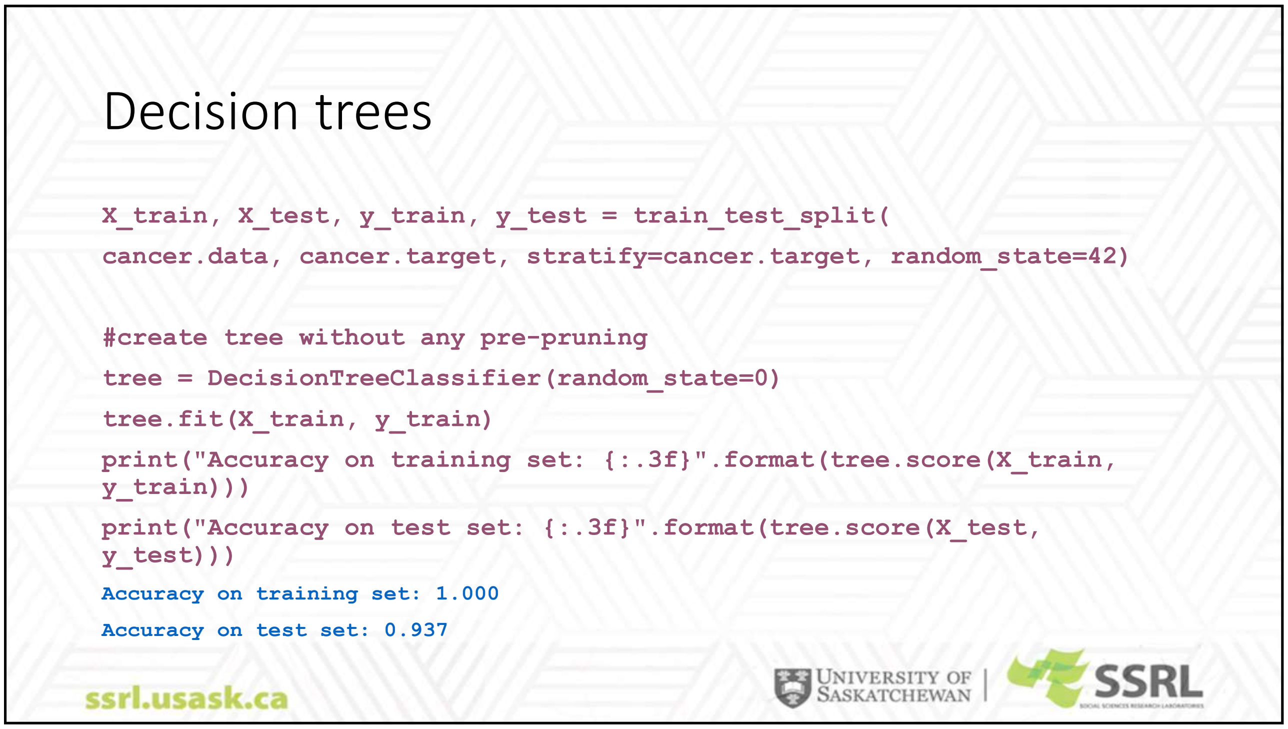
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer = load_breast_cancer()
```



Decision trees

```
#569 instances, 30 features
print("Target names:", cancer['target_names'])
Target names: ['malignant' 'benign']

print("Feature names:\n", cancer['feature_names'])
Feature names:
 ['mean radius' 'mean texture' 'mean perimeter' 'mean area'
'mean smoothness' 'mean compactness' 'mean concavity'
'mean concave points' 'mean symmetry' 'mean fractal dimension'
'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```

ssrl.usask.ca University of Saskatchewan | SSRL



Decision trees

```
X_train, X_test, y_train, y_test = train_test_split(
cancer.data, cancer.target, stratify=cancer.target, random_state=42)

#create tree without any pre-pruning
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree.score(X_train,
y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test,
y_test)))
Accuracy on training set: 1.000
Accuracy on test set: 0.937
```

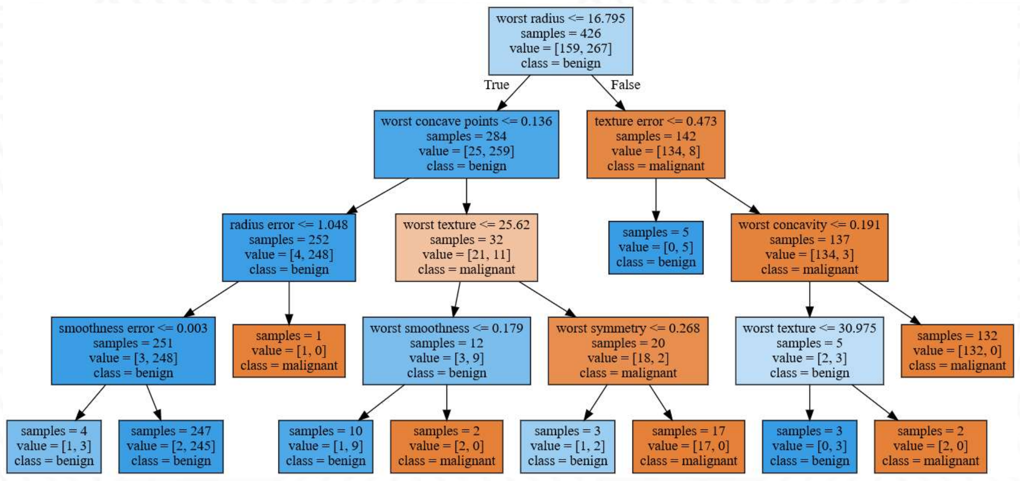
ssrl.usask.ca University of Saskatchewan | SSRL

Decision trees

```
#create tree by limiting it to max-depth of 4
tree = DecisionTreeClassifier(max_depth=4, random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree.score(X_train,
y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test,
y_test)))
```

```
Accuracy on training set: 0.988
Accuracy on test set: 0.951
```

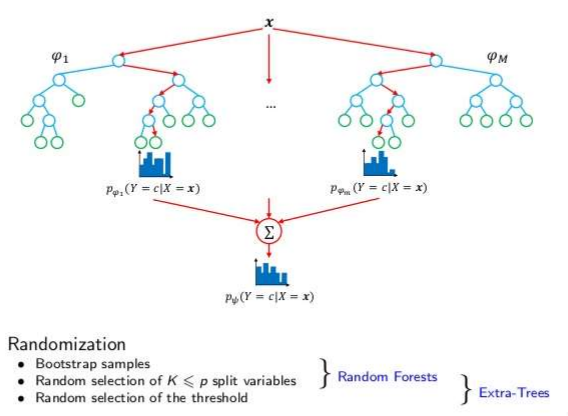
Decision trees

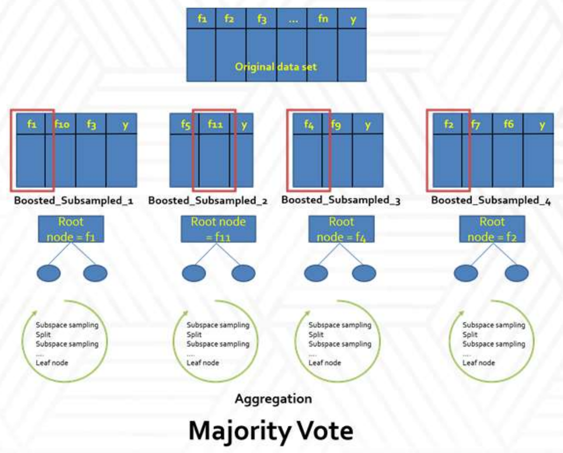


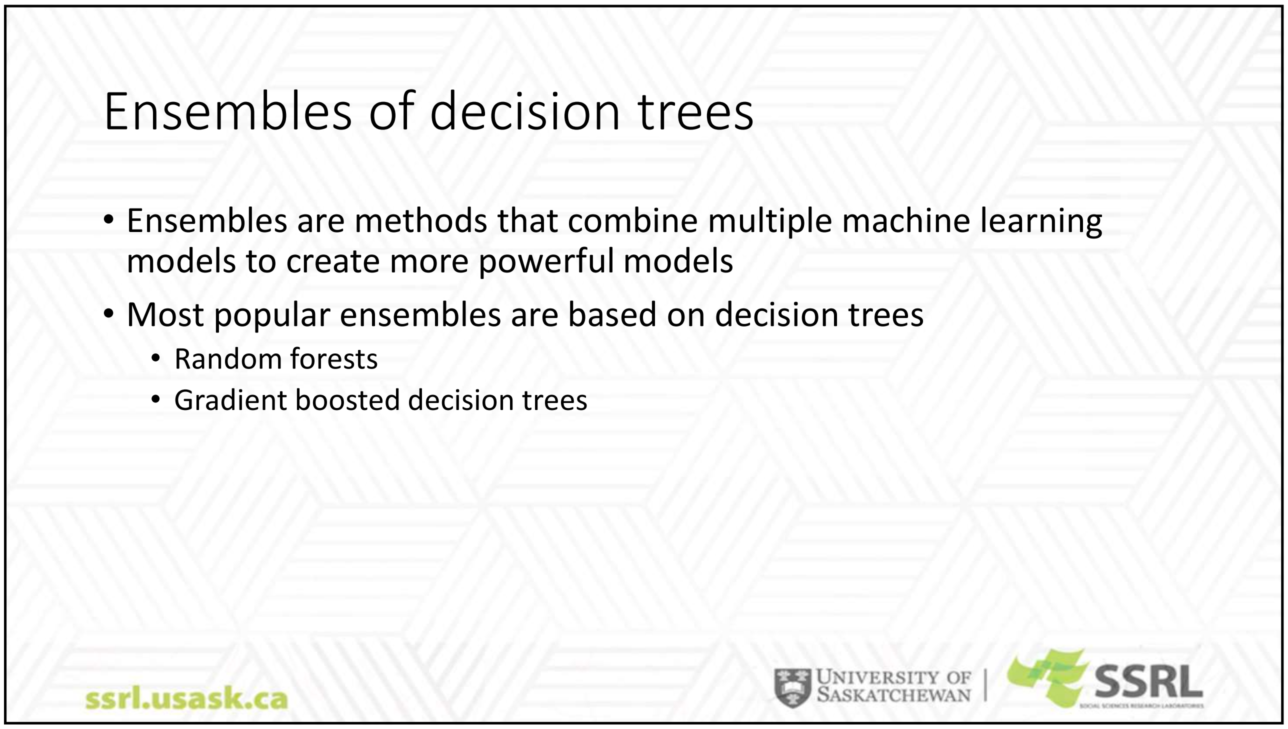
Decision trees

- Advantages
  - Model can be easily visualized and understood by non-experts
  - No preprocessing of the data is needed
- Disadvantage
  - Even with pre-pruning, decision trees tend to overfit and provide poor generalization performance

Random forests

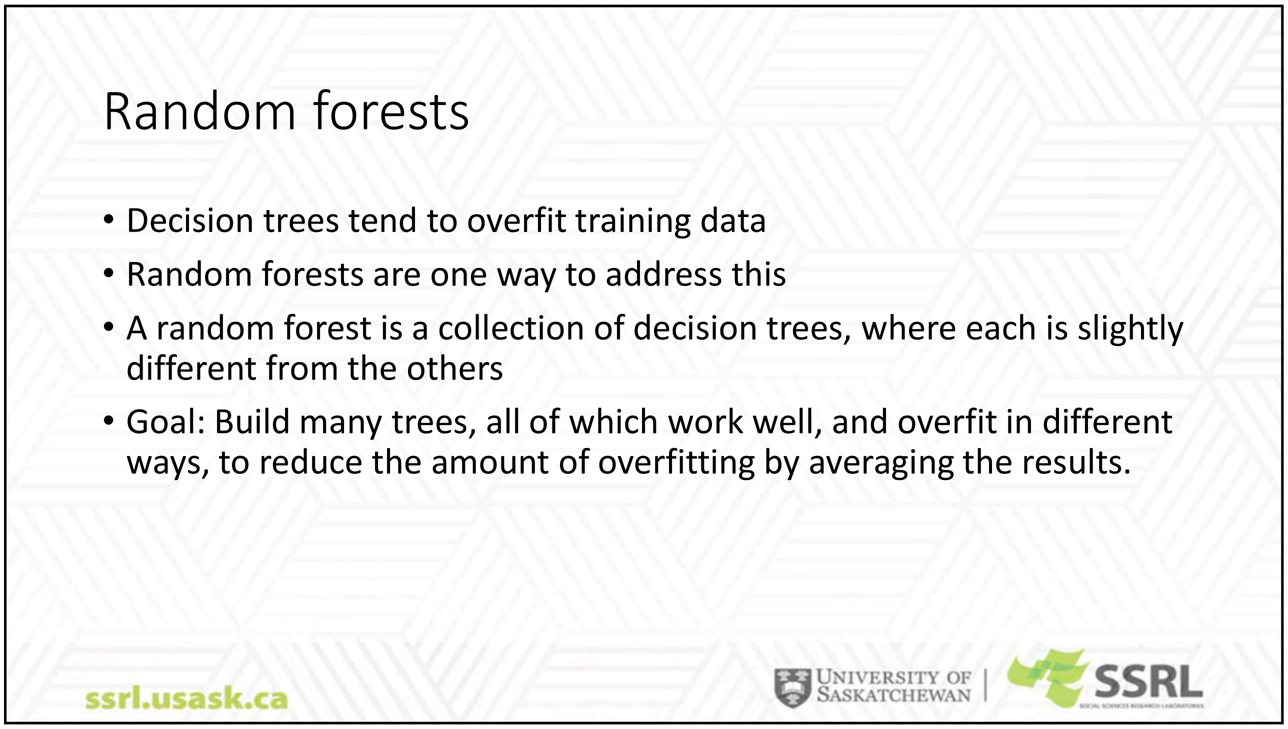

x
φ_1
φ_M
...
$p_{\varphi_1}(Y = c|X = x)$
$p_{\varphi_m}(Y = c|X = x)$
Σ
$p_\psi(Y = c|X = x)$
Randomization
Bootstrap samples
Random selection of $K \leqslant p$ split variables
Random selection of the threshold
Random Forests
Extra-Trees





Ensembles of decision trees

- Ensembles are methods that combine multiple machine learning models to create more powerful models
- Most popular ensembles are based on decision trees
  - Random forests
  - Gradient boosted decision trees



Random forests

- Decision trees tend to overfit training data
- Random forests are one way to address this
- A random forest is a collection of decision trees, where each is slightly different from the others
- Goal: Build many trees, all of which work well, and overfit in different ways, to reduce the amount of overfitting by averaging the results.

Random forests

- Need to build many trees. Each should do an acceptable job of predicting the target.
- Random forests get their name from injecting randomness into tree building to ensure each tree is different.
- Two ways trees are randomized
  - By randomly selecting the data points used to build a tree
  - By randomly selecting the features in each split test

Random forests

- Need to decide how many trees to build (the **n_estimators** parameter in Scikit-learn)
- For each tree in the forest, take a *bootstrap sample* of the data.
  - From n_samples data points, repeatedly draw randomly <u>with replacement</u>, n_samples times
  - This will create a dataset that is as big as the original dataset, but some data points will be missing (approx. 1/3), and some will be repeated

Random forests

- For each bootstrap sample, grow an unpruned tree
  - For each decision node, randomly selects a subset of features, and determine the split using only those features
  - The number of feature that are selected is controlled by the **max_features** parameter in Scikit-learn
  - The algorithm continues to randomly select features for each node, so that each node can make decisions using a different subset of features

Random forest algorithm

Let N_{trees} by the number of trees to build

For each of the N_{trees} iterations

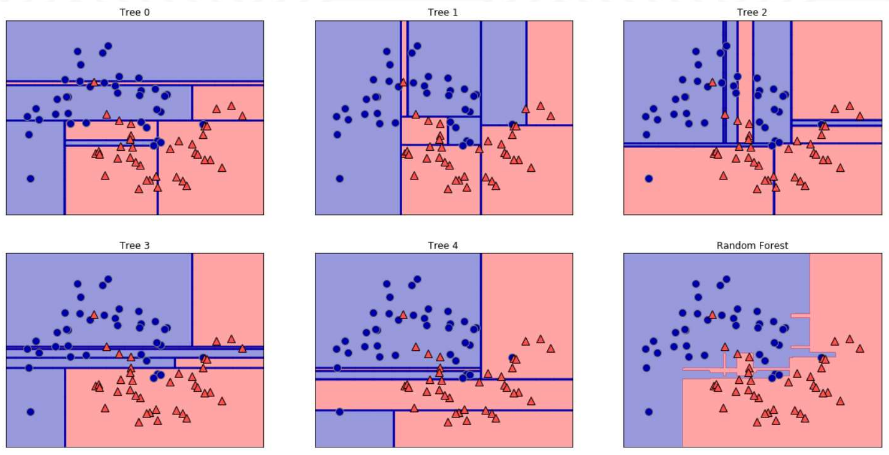
1. Select a new bootstrap sample from the training set
2. Grow an un-pruned tree on this bootstrap
3. At each internal node, randomly select m_{try} features and determine the split using only these features
4. Do not perform cost complexity pruning. Save the tree as is, along side those build so far

Random forests

- To make a prediction using the random forest, the algorithm first makes a prediction for every tree in the forest.
- For regression, it averages the results to get the final prediction
- For classification, the algorithm provides the probabilities of each possible output label, and the majority vote from the trees is used to make the decision.

ssrl.usask.ca

Random forests



ssrl.usask.ca

Random forests

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
```

UNIVERSITY OF SASKATCHEWAN | SSRL

Random forests

```
X_train, X_test, y_train, y_test = train_test_split(
    cancer.data, cancer.target, random_state=0)

forest = RandomForestClassifier(n_estimators=100, random_state=0)
forest.fit(X_train, y_train)

print("Accuracy on training set: {:.3f}".format(forest.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(forest.score(X_test, y_test)))
Accuracy on training set: 1.000
Accuracy on test set: 0.972
```

UNIVERSITY OF SASKATCHEWAN | SSRL

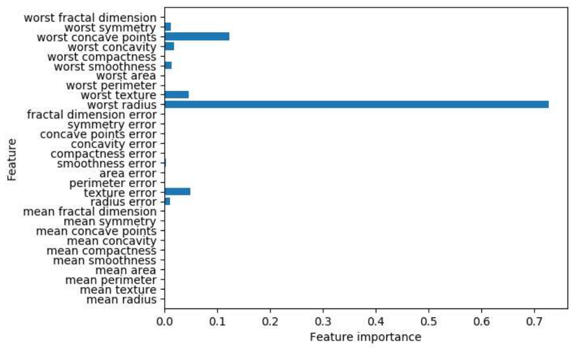
Decision tree vs. random forest

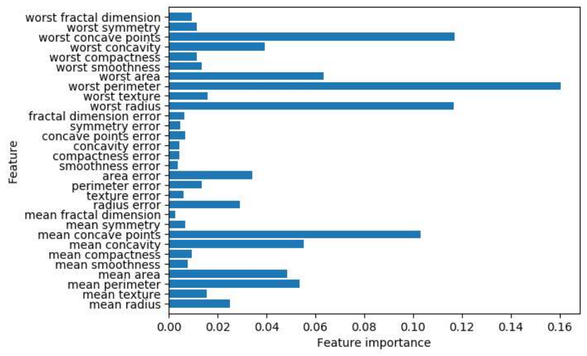
- Wisconsin breast cancer diagnostic dataset
- Decision Tree
  - `Accuracy on training set: 0.988`
  - `Accuracy on test set: 0.951`
- Random Forest
  - `Accuracy on training set: 1.000`
  - `Accuracy on test set: 0.972`

Random forests

```
def plot_feature_importances_cancer(model):
    n_features = cancer.data.shape[1]
    plt.barh(np.arange(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), cancer.feature_names)
    plt.xlabel("Feature importance")
    plt.ylabel("Feature")
    plt.ylim(-1, n_features)

plot_feature_importances_cancer(forest)
```

Decision tree vs. random forest





ssrl.usask.ca

University of Saskatchewan | SSRL Social Sciences Research Laboratories

Random forests

- Advantages
  - Random forests for regression and classification are among the most widely used machine learning methods
  - It is one of the more accurate learning algorithms available. For many data sets, it produces a highly accurate result.
  - It works well for dense data (<= a few thousand features)
  - It often work well without heavy tuning of parameters, and doesn’t require data scaling.
  - It gives estimates of what variables are important in the classification.

ssrl.usask.ca

University of Saskatchewan | SSRL Social Sciences Research Laboratories

Random forests

- Disadvantages
  - Other algorithms may have better accuracy. Deep neural networks generally do better, and sometimes gradient boosted trees.
  - Random forests are random, and setting different random states can drastically change the model that is built (often mitigated when large forests are used).

Gradient boosted regression trees

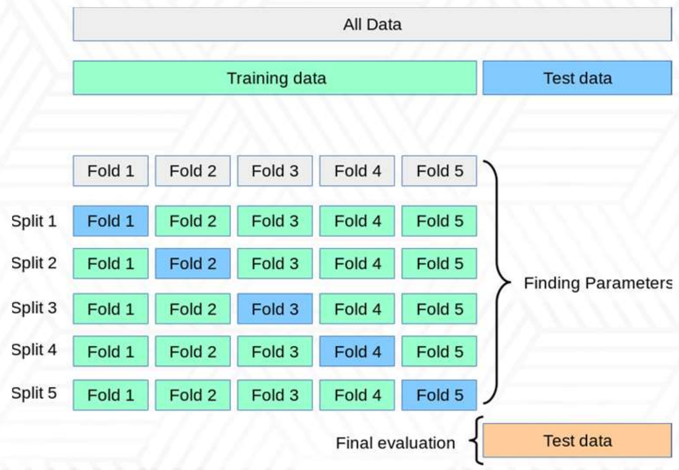
- Another ensemble method that combines multiple decision trees
- Can be used for regression or classification
- Builds trees in serial manner, where each tree tries to correct the errors of the previous one (RF builds in parallel)
- No randomness. Pre-pruning is used.
- Combines many shallow trees, and each tree provides good coverage of part of the data

GBRT vs. Random Forest

- Random forest (RF) has about the same accuracy as GBRT for classification.
- For continuous (regression), GBRT appears to outperform RF
- GBRT requires more attention to parameter tuning than RF
- RF generally does not overfit, but GBRF can if the parameter are not managed properly
- RF performance is much better on hardware that handles parallel processing (i.e. multiple processors)

ssrl.usask.ca

K-fold cross validation



ssrl.usask.ca

Cross validation

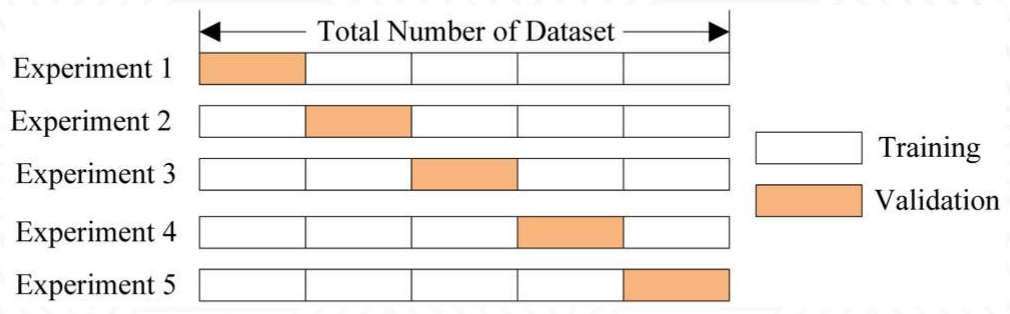
- Test results may be invalidated when practitioners inadvertently peek at the test data.
- When parameters in models are tweaked to make sure that the test data performs optimally, overfitting occurs on the test set
- When this occurs, the model and evaluation metrics no longer report on generalization performance
- To solve this problem, another part of the training dataset can be held out as a *validation set*
- Training proceeds on the training set, and evaluation is done on the validation set
- *Validation results are used to tune the model parameters* until validation testing seems successful, and then final evaluation is done on the test set.

K-fold cross validation algorithm

- A common and simple cross-validation technique

- Shuffle the training dataset randomly.
- Split the training dataset into k groups / folds (usually 5 or 10)
- For each unique group:
  - Take a group as a validation or test data set
  - Take the remaining groups as a training data set
  - Fit a model on the training set and evaluate it on the validation/ test set
  - Retain the evaluation score and discard the model
- Use the average of the evaluation scores from the folds as the metric
- Perform a final evaluation using the “real” test set

K-fold cross validation



ssrl.usask.ca

University of Saskatchewan | SSRL

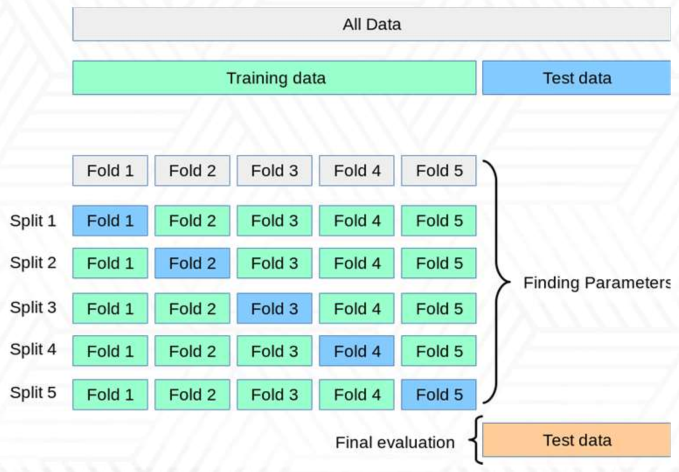
K-fold cross validation

```
from numpy import array
from sklearn.model_selection import KFold

data = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])
# prepare cross validation
kfold = KFold(3, True, 1)
# enumerate splits
for train, test in kfold.split(data):
      print('train: %s, test: %s' % (data[train], data[test]))

train: [0.1 0.4 0.5 0.6], test: [0.2 0.3]
train: [0.2 0.3 0.4 0.6], test: [0.1 0.5]
train: [0.1 0.2 0.3 0.5], test: [0.4 0.6]
```

ssrl.usask.ca

University of Saskatchewan | SSRL

K-fold cross validation



ssrl.usask.ca

University of Saskatchewan | SSRL

Summary: Algorithms to know

- Supervised learning
  - KNN (K-Nearest neighbor)
  - Logistic regression
  - Linear regression
  - Decision tree
  - Random forest
  - Gradient boosting regression trees
  - Support sector machines(SVM)
  - Naive Bayes
- Unsupervised learning
  - K-Means
- Dimension reduction
  - Principal component analysis

ssrl.usask.ca

University of Saskatchewan | SSRL

Summary: Algorithms to know

- Supervised learning
  - **KNN (K-Nearest neighbor)**
  - **Logistic regression**
  - **Linear regression**
  - **Decision tree**
  - **Random forest**
  - **Gradient boosting regression trees**
  - Support sector machines(SVM)
  - Naive Bayes
- Unsupervised learning
  - K-Means
- Dimension reduction
  - Principal component analysis

Machine learning steps

- Gathering data
- Preparing data
- Model building
- Parameter tuning
- Evaluation
- Prediction

Machine learning steps

- Gathering data
- Preparing data
- **Model building**
  - Build models that are appropriate for dataset, problem, task
  - Build multiple models (no free lunch theorem)
- Parameter tuning
- Evaluation
- Prediction

ssrl.usask.ca

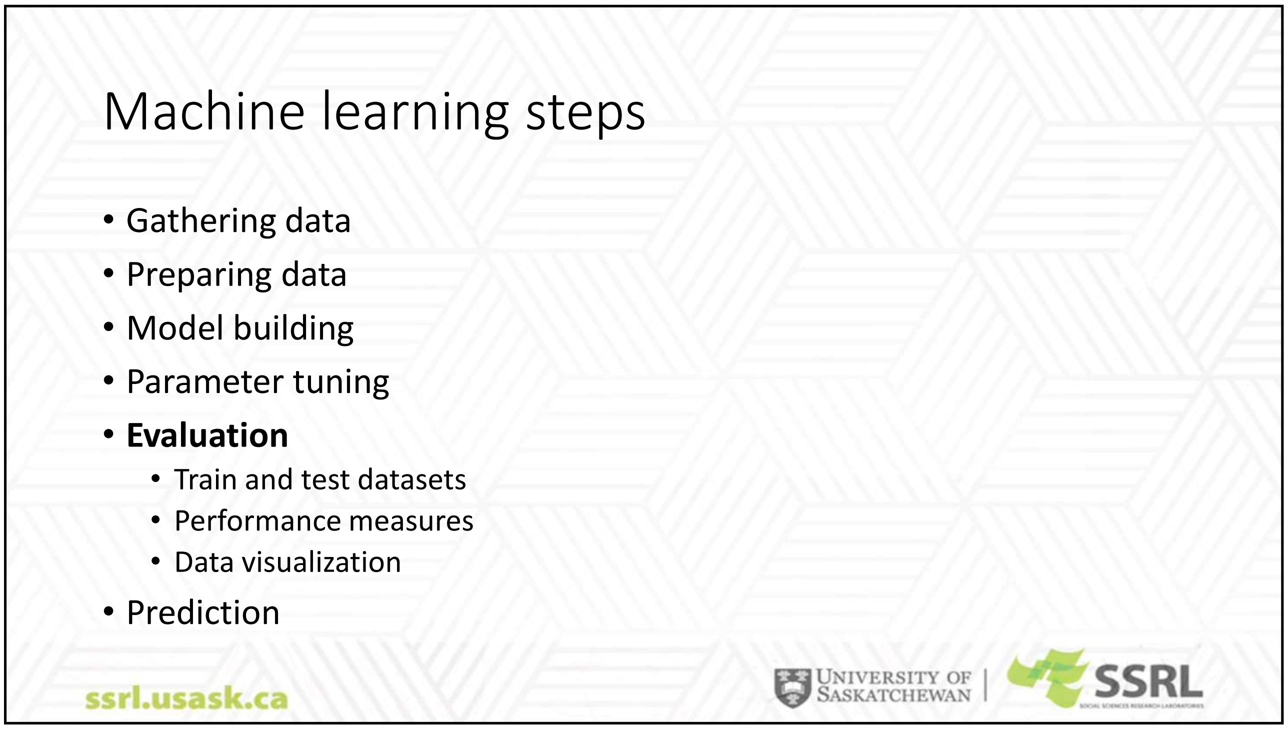
UNIVERSITY OF SASKATCHEWAN | SSRL

Machine learning steps

- Gathering data
- Preparing data
- Model building
- **Parameter tuning**
  - Cross-validation (K-fold cross validation)
  - Data visualization
- Evaluation
- Prediction

ssrl.usask.ca

UNIVERSITY OF SASKATCHEWAN | SSRL

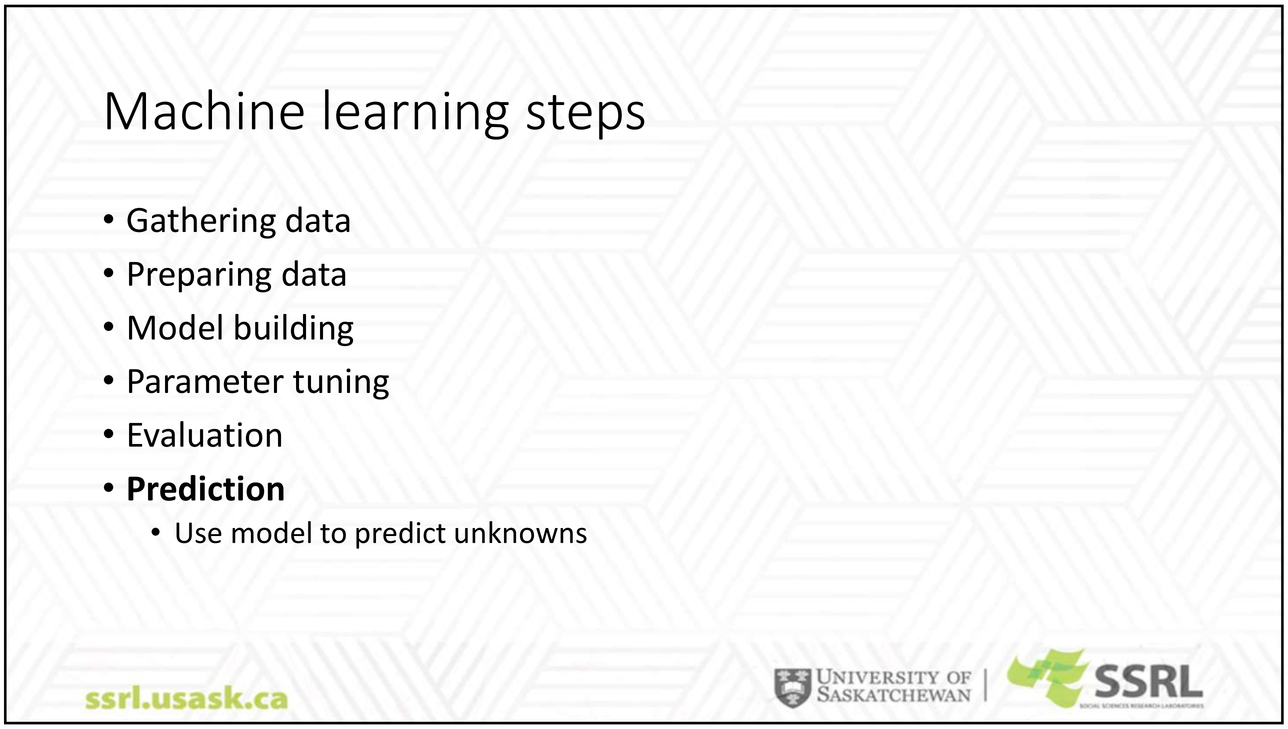


Machine learning steps

- Gathering data
- Preparing data
- Model building
- Parameter tuning
- **Evaluation**
  - Train and test datasets
  - Performance measures
  - Data visualization
- Prediction

ssrl.usask.ca

University of Saskatchewan | SSRL Social Sciences Research Laboratories

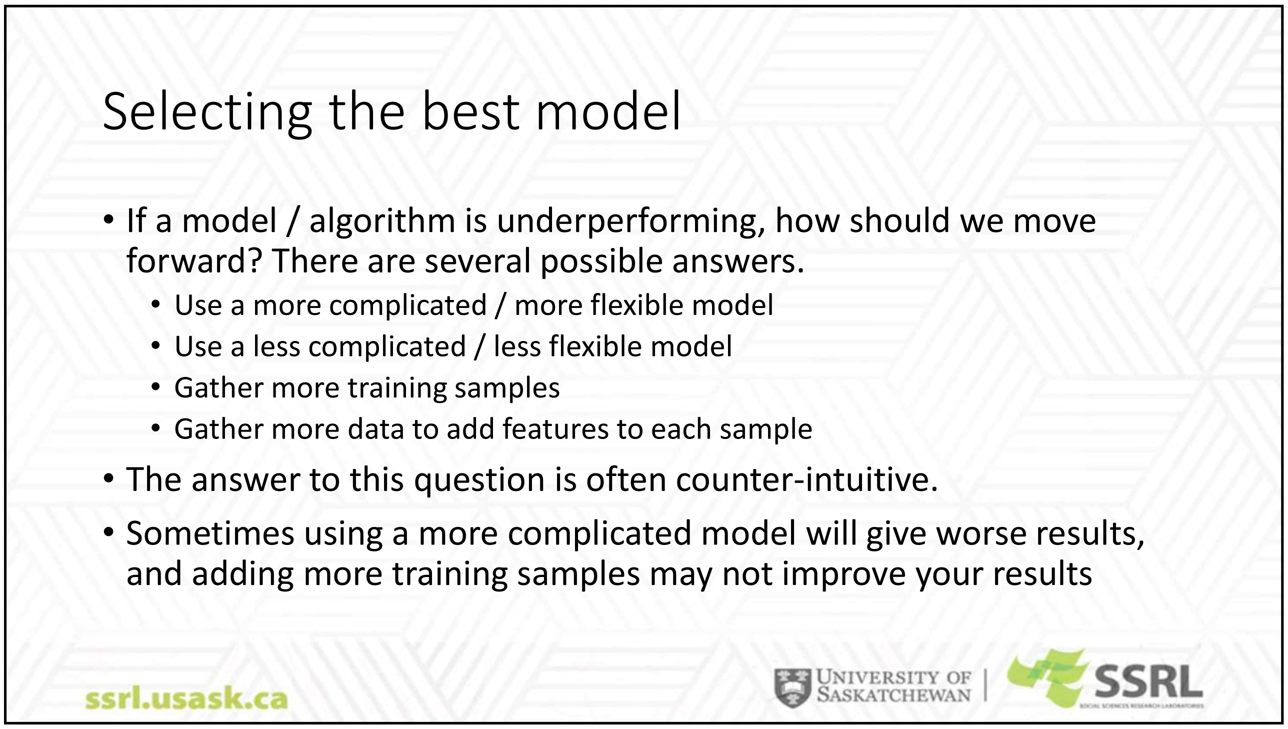


Machine learning steps

- Gathering data
- Preparing data
- Model building
- Parameter tuning
- Evaluation
- **Prediction**
  - Use model to predict unknowns

ssrl.usask.ca

University of Saskatchewan | SSRL Social Sciences Research Laboratories

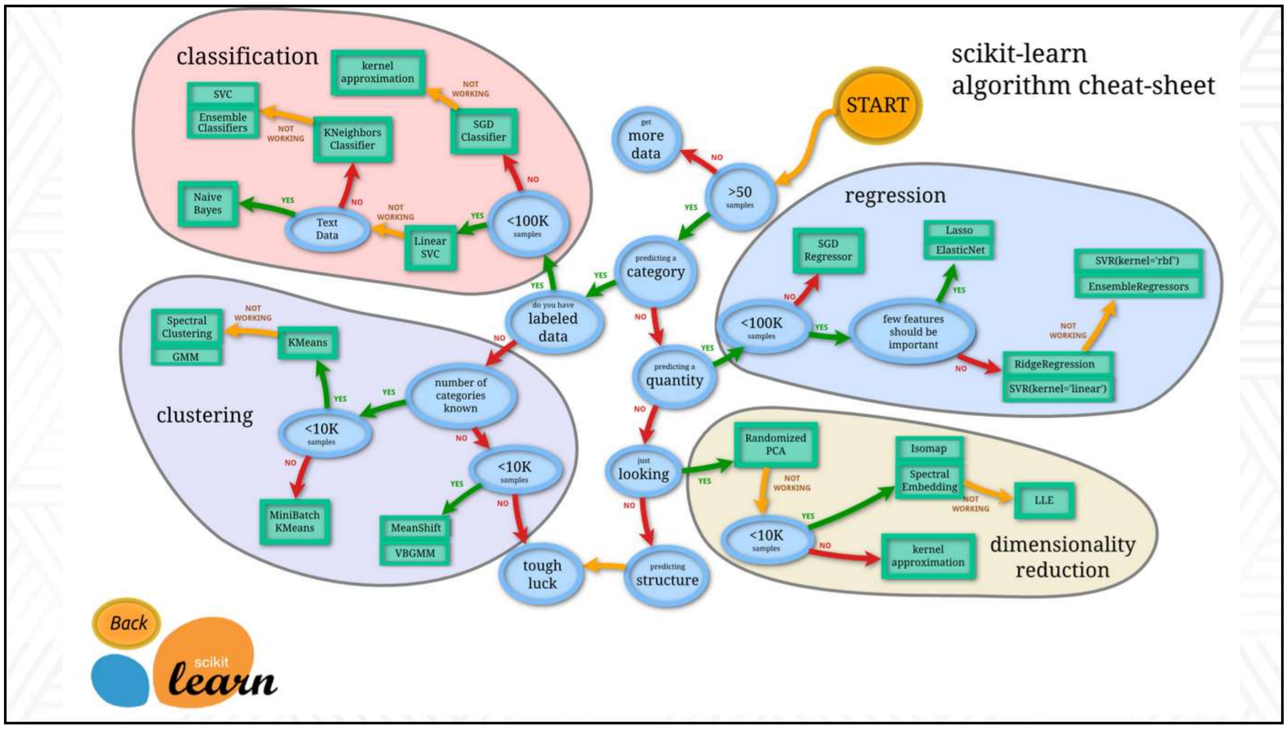


Selecting the best model

- If a model / algorithm is underperforming, how should we move forward? There are several possible answers.
  - Use a more complicated / more flexible model
  - Use a less complicated / less flexible model
  - Gather more training samples
  - Gather more data to add features to each sample
- The answer to this question is often counter-intuitive.
- Sometimes using a more complicated model will give worse results, and adding more training samples may not improve your results

ssrl.usask.ca

UNIVERSITY OF SASKATCHEWAN | SSRL SOCIAL SCIENCES RESEARCH LABORATORIES



Machine learning toolkits

- **“TensorFlow:** With the initial release of this open source machine learning framework being 2015, it has been deployed across many different platforms and is easy to use. Created by Google at first, now all the top tech giants such as eBay, Dropbox, Intel and Uber use it extensively. With the help of flowgraphs, one can develop neural networks.”
- **“Microsoft Cognitive Toolkit:** Initially released about three years back, this is an AI solution that you can use to take your machine learning projects to the next level in every way. Certain studies have revealed that the open source framework can train certain algorithms to function like the human brain.”

https://www.analyticsinsight.net/the-6-most-important-ai-technologies-in-machine-learning/

ssrl.usask.ca

Machine learning as a service

- Amazon web services
  - “81% of deep learning projects in the cloud run on AWS”
  - “85% of TensorFlow projects in the cloud run on AWS”
- Google Cloud AI hub
  - “Hosted repository of plug-and-play AI components”
  - Supported toolkits
    - TensorFlow
    - Scikit-learn
    - PyTorch
    - Keras
- Microsoft Azure
  - Microsoft Azure Machine Learning Studio uses “drag and drop machine learning”
  - Allows models to be published as a web service
  - **Microsoft’s Azure Cognitive Services adds reinforcement learning for recommendations and doodle recognition AI (**May 2, 2019)

ssrl.usask.ca

Machine learning news

- **Facebook open-sources deep learning framework Pythia for image and language models (**May 21, 2019)
- **Google releases four new machine learning APIs for developers** (May 21, 2019)
- **Microsoft open sources algorithm that gives Bing some of its smarts** (May 15, 2019)