Machine Learning for Finance

Machine Learning

Dr Ning Wang

2019

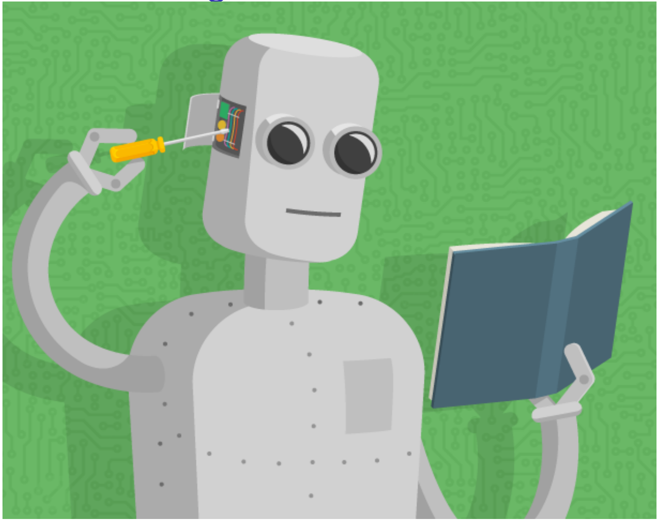
Outline

- What is Machine Learning?
- Cross-validation
- Regressions (linear and logistic)
- Classification Tree
- Support Vector Machine
- Neural Networks
- Deep Neural Learning

Literatures

- *The Elements of Statistical Learning* by T. Hastie, R. Tibshirani and J. Friedman, Second edition, Springer, 2009
- *Deep Learning* by Ian Goodfellow and Yoshua Bengio, 2016

Machine Learning

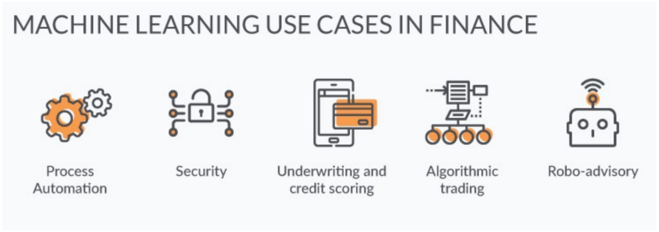


“Scientific study of algorithms and statistical models that computer systems use to progressively improve their performance on a specific task.” - Wikipedia

Definition

- Machine learning seeks to answer the question:
  - "How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes?" - Tom Mitchell
- Machine learns with respect to a particular task T, performance metric P, and type of experience E
  - A computer learns if it improves its performance P at some task T with experience E (i.e. more data)
  - Extracting a model of a system from the observations (or the simulations) in some situations
  - The model presents some relationships between the variables used to describe the system

Machine learning for Finance


MACHINE LEARNING USE CASES IN FINANCE
Process Automation
Security
Underwriting and credit scoring
Algorithmic trading
Robo-advisory

- Reduced operational costs through process automation
- Increased revenues due to better productivity and enhanced user experiences
- Better compliance and reinforced security

Machine Learning in Finance

- Humans have shown not very much skill at making superior investments
  - 86% money managers failed to beat their benchmarks [1]
- Predictive analytics for finance (ex. fraud detection, credit risk)
- Analysis of invest's sentiment on social media streams
- Quants and algotrading - patterns in the data
- Better understand systemic risk in financial systems
- Better understand patterns in investor and consumer behavior
- Opportunities of digital currencies and cryptography

[1] http://money.cnn.com/2015/03/12/investing/investing-active-versus-passive-funds

Financial data

- Public data
  - Stock, Forex, Oil . . .
  - Google, Yahoo . . .
  - Gov, Open data . . .
- Semi-public data
  - data vendors (stocks, bonds, funds, options, futures . . .)
  - media (Bloomberg, Reuters . . .)
  - API (Quandl)
- Internal data
  - valuation information, fundamental data, reference data . . .
- Unstructured data
  - media reports (FT, . . .)
  - social media (Twitter, FB . . .)

Financial Data Analytics

- Financial Data Analytics can be broken down into a series of steps:
  1. Data collection (stucture and unstructure data)
  2. Data preprocessing and preparation (cleaning; inconsistant data; spliting data; 80/20 or 70/30 rules)
  3. Data exploration (learning more about the data and its nuances)
  4. Data modeling (regression; machine learning)
  5. Model evaluation (biased results; performance criteria; benchmark; testing data; out of sample)
  6. Performance improvement (more data; better model; parallel computing; ensembling; scalibility)

Cross-validation

- Learning and testing the model on the same data is a methodological mistake
- Cross-validation aims to detecte and prevent overfitting
- Accuracy on the training set is optimistic
- A better estimate comes from an independent set (*testing* set)
- We can't use the *testing* set when building the model or it becomes part of the training set
- We estimate the *testing* set accuracy with the *training* set

Cross-validation

- The model $\hat{f}_\lambda$ depends on a parameter λ
- We aim to identify the model that gives the *best result* (see below) *on the whole population* (not the one that gives the *best* result on the *testing set*)
- Given the training set $\mathcal{T}$, we aim to minimise the *generalisation error*:

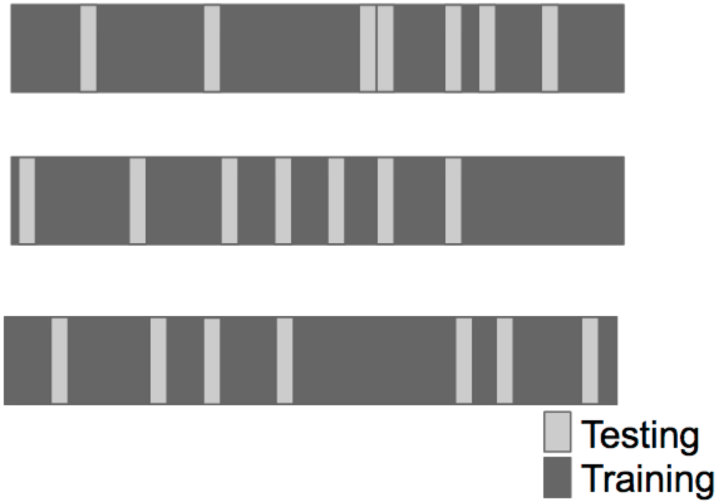
  $$\min_\lambda \mathbb{E}\left[L(y, \hat{f}_\lambda(x)) | \mathcal{T}\right]$$

  where: L is the loss function (e.g. $L(u, v) = (u - v)^2$),
- Alternatively, minimise the *prediction error*:

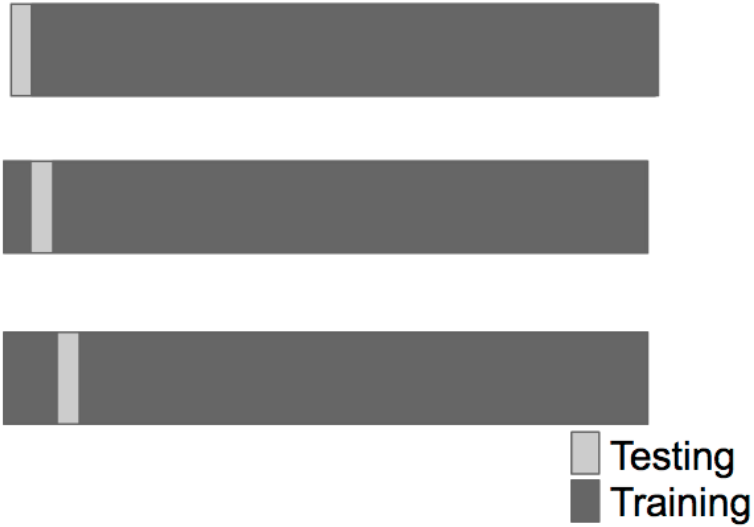
  $$\min_\lambda \mathbb{E}\left[L(y, \hat{f}_\lambda(x))\right]$$

- The expectation is to be estimated on a large enough *validation* set that is independent from the actual training set.

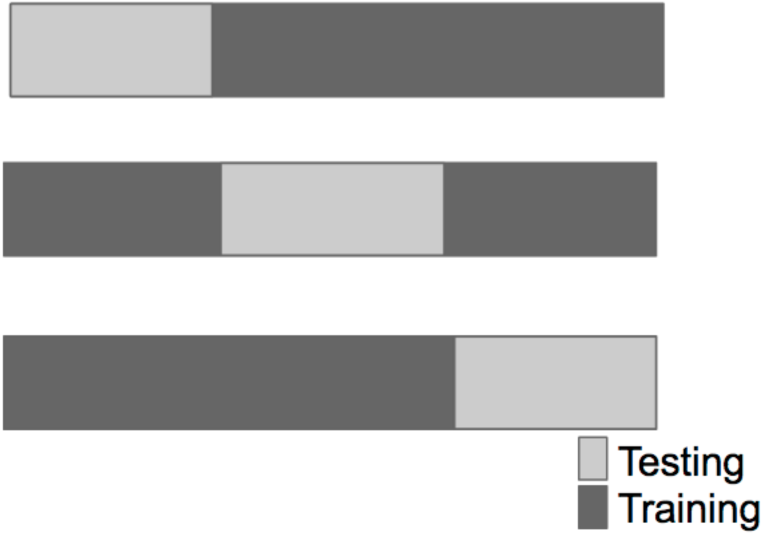
Random subsampling



Leave one out



K-fold



Cross-validation considerations

- For time series data data must be used in “chunks”
- Random sampling must be done *without replacement*
- Random sampling with replacement is the *bootstrap*
  - Underestimates of the error
  - Can be corrected, but it is complicated (http://www.jstor.org/discover/10.2307/2965703?uid=2&uid=4&sid=21103054448997)
- For k-fold cross validation
  - Large k: approximately unbiased predictor, but high variance and potentially high computational cost
  - Low k: biased predictor, but low variance and lower computational cost

Data splitting functions in Python

- *Scikit-learn* performs a (supervised) machine learning experiment to hold out part of the available data as a test set X_test, y_test.
  - `train_test_split`, `KFold`, `LeaveOneOut`, `TimeSeriesSplit`

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([1, 2, 3, 4])
X_train, X_test, y_train, y_test = train_test_split(X, y,
  test_size=0.5, random_state=0)
print(X_train.shape, X_test.shape)
```

```
## ((2, 2), (2, 2))
```

Data splitting functions in Python

```python
# http://scikit-learn.org/stable/modules/cross_validation.html
kf = KFold(n_splits=2, random_state=None, shuffle=False) # Define 2 folds
kf.get_n_splits(X) # returns the number of splitting iterations
for train_index, test_index in kf.split(X):
  print("TRAIN:", train_index, "TEST:", test_index)
```

```
## ('TRAIN:', array([2, 3]), 'TEST:', array([0, 1]))
## ('TRAIN:', array([0, 1]), 'TEST:', array([2, 3]))
```

Choosing R or Python for data analysis?

R and Python are waging war:
while both programming languages are gaining prominence
in the data analytics community, they are fighting
to become data scientists' language of choice.

Which side are you taking?

Choosing R or Python for data analysis?

- It's up to you to choose
  - What problems do you have to solve?
  - Which language best fits your needs?
  - What are the costs to learn the language?
  - What is your community?

Machine Learning for Finance

Regularised Regression

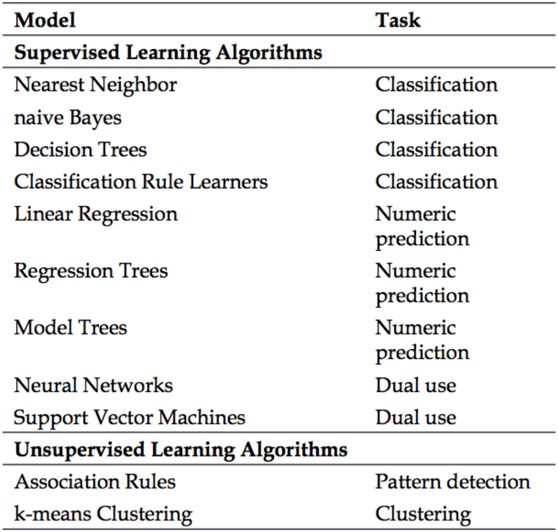
Dr Ning Wang

2019

This lecture

- Brief overview of supervised learning (HTF 2, 7.2)
- Regularised regression (HTF 3.4)
- Ridge regression (HTF 3.4.1)
- Lasso shrinkage (HTF 3.4.2, 3.4.3)

General types of machine learning algorithms



Model	**Task**
Supervised Learning Algorithms	
Nearest Neighbor	Classification
naive Bayes	Classification
Decision Trees	Classification
Classification Rule Learners	Classification
Linear Regression	Numeric prediction
Regression Trees	Numeric prediction
Model Trees	Numeric prediction
Neural Networks	Dual use
Support Vector Machines	Dual use
Unsupervised Learning Algorithms	
Association Rules	Pattern detection
k-means Clustering	Clustering

Supervised learning

- Goal: from the database (learning sample), find a function h of the inputs that approximates at best the output
  - Numerical output: regression problem
  - Symbolic output: classification problem
  - Predictive: make predictions for a new sample described by its attributes
- Supervised learning model
  - given x *input*, *features*, *predictors*, *independent variables*
  - and corresponding y *output*, *response*, *dependent variables*,
  - we assume that y are observed (noisy) values of $f(x)$ for some f,
  - we aim to estimate f by $\hat{f}$.
- Financial Applications: time series analysis; fraud detection; credit scoring; etc

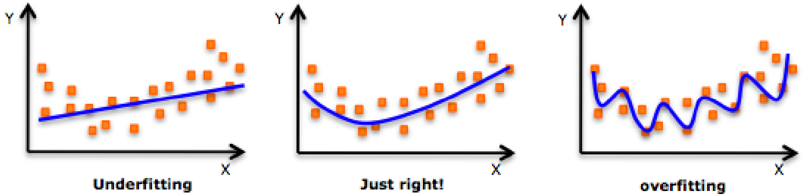
Supervised learning process (ref. HTF 7.2)

- Define the learning problem
- Generate or collect the data
- Data preprocessing and splitting
  - Split the data into independent *training* and *testing* sets
- Train the model
  - Fit the model on the *training* set
- Evaluate the performance
  - Valiate the model on the testing set (estimate the *prediction error*)
  - If not satisfied go back to *training*
  - If the model is chosen out of a few, compare them on a *validation* set, that is independent from the *training* set, in order to *select* the *best* one,
  - Assess the final model (estimate the *prediction error*) on a *testing set*.

Evaluating the success of learning

- The model is tested on testing data
  - How well its characterization of the training data generalizes to the testing data
- Every learner has its weaknesses and biases
  - Bias is associated with the abstraction and generalization process
- Failure to generalize usually is caused by noisy data

Overfitting



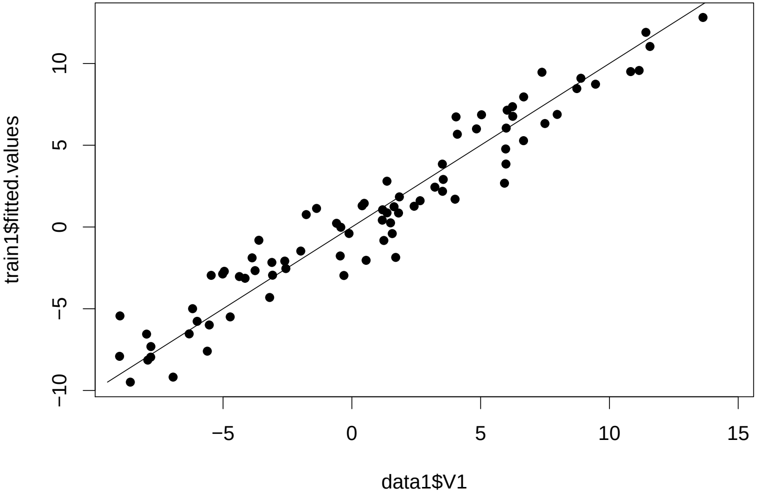
- Overfitting means the model does not generalize well
  - The model performs well during training but does poorly during testing overfitted to the training dataset
- Overfitting models the noise in data
  - Noise is random by definition, attempting to explain the noise will result in erroneous conclusions
- Overfitting results in more complex models that are more likely to ignore the true pattern
- Solutions to overfitting are specific to particular machine learning approaches

Linear supervised learning

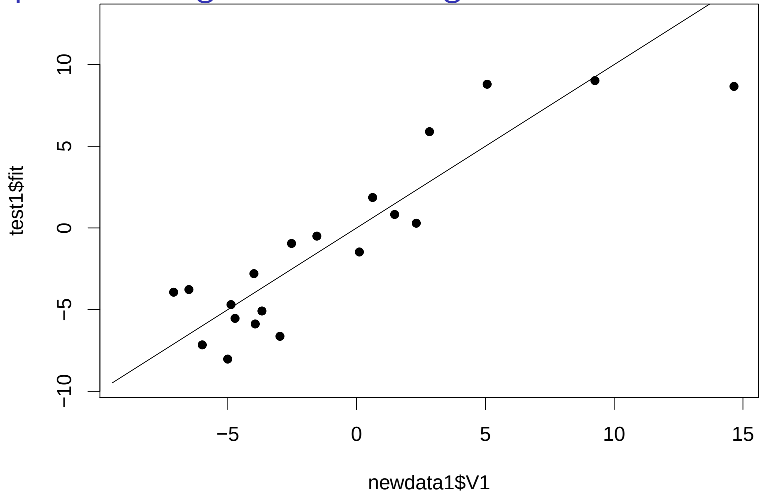
- Many real problems can be approximated with linear models.
- Linear prediction provides an example to many of the core concepts of machine learning.

Simple linear regression - training

First we generate some toy data in order to demonstrate the need for regularisation. The residual standard error on the training set is 1.9211855. The regression is fairly accurate.



Simple linear regression - testing



- The sum of squares of the residuals on the testing set is 120.6703309.
- The estimates on the testing set are reasonably accurate.

Making the data noisier

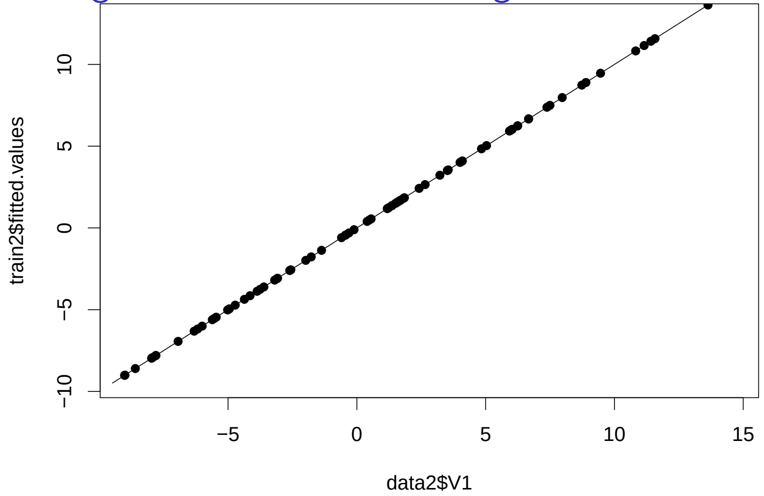
Now, we add some extra features in order to confuse the traditional linear regression.

This mimics more realistic situations:

- the feature set is rich enough to contain the required information
- but in contains information that may be redundant or may not be relevant

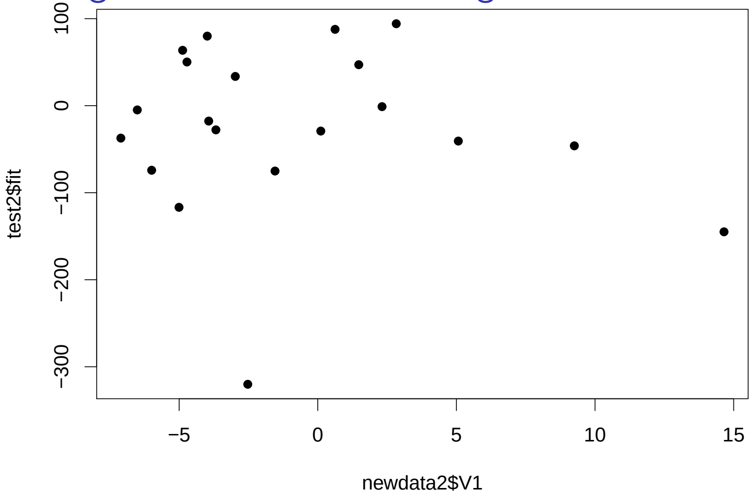
On the next slides we demonstrate why it is problematic.

Linear regression revisited - training



- The sum of squares of the residuals is on the training set is NaN.
- This seems to be a perfect fit.

Linear regression revisited - testing



- The sum of squares of the residuals on the testing set is 190749 (compare to `err1.te` above).
- The estimator is over-fitted to the training set.

Regularised regression - background

Regularised regression models impose penalty on the regression coefficients:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|^q \right\}$$

Where:

- the $q = 2$ case is referred to as Ridge regression
- the $q = 1$ case is called Lasso (least absolute shrinkage and selection operator) method

Alternatively, one can represent the optimisation problem as:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\beta_j)^2 \right\}$$

$$\text{subject to } \sum_{j=1}^{p} |\beta_j|^q \leq t.$$

Ridge regression - derivation

The coefficients of the Ridge regression ($q = 2$) can be directly computed.

- Step 1. $\beta_1 = \frac{1}{N} y_i = \bar{y}$.
- Step 2. Introduce the notation:

$$\beta = (\beta_i)_{j=1}^{d}, \ \mathbf{y} = (y_i - \bar{y})_{i=1}^{N}, \ \mathbf{x} = (x)_{i,j}$$

Then, the objective function can be written as

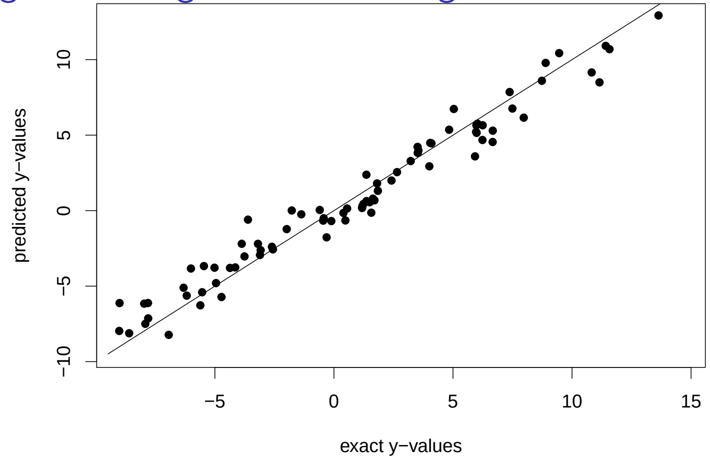
$$(\mathbf{y} - \mathbf{x}\beta)^T(\mathbf{y} - \mathbf{x}\beta) + \lambda\beta^T\beta,$$

and

$$\hat{\beta} = (\mathbf{x}^T\mathbf{x} + \lambda\mathbf{I})^{-\mathbf{1}}\mathbf{x}^{\mathbf{T}}\mathbf{y},$$

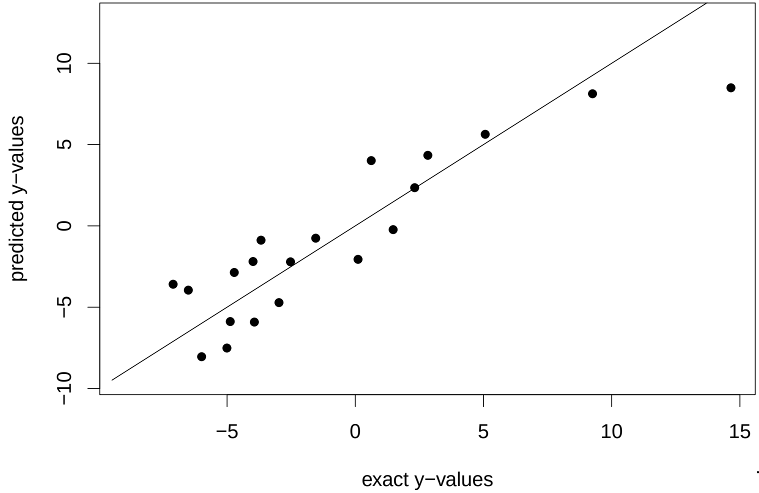
where $\mathbf{I}$ is the $d \times d$ identity matrix.

Regularised regression - training with Lasso



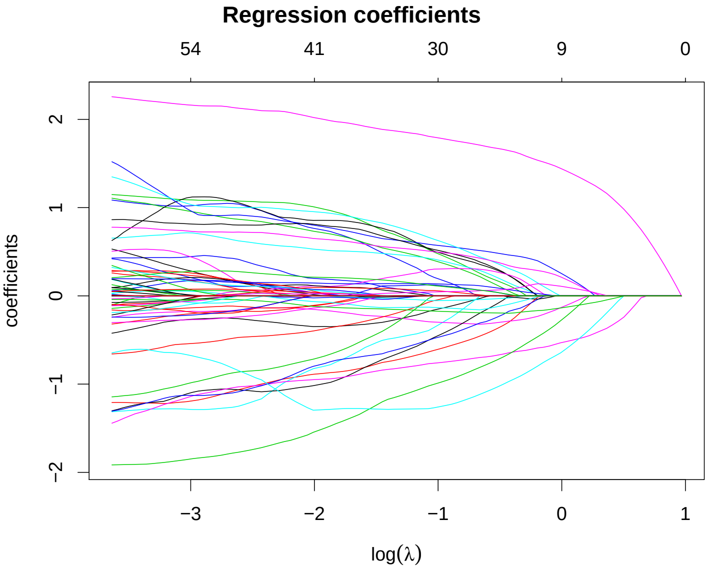
- The perfect fit in the previous case used most of the variables (see ‘summary(train2)’)
- In the case of Lasso regularisation, when most of the coefficients are used, the penalty term blows up (see later).

Regularised regression - Lasso, testing

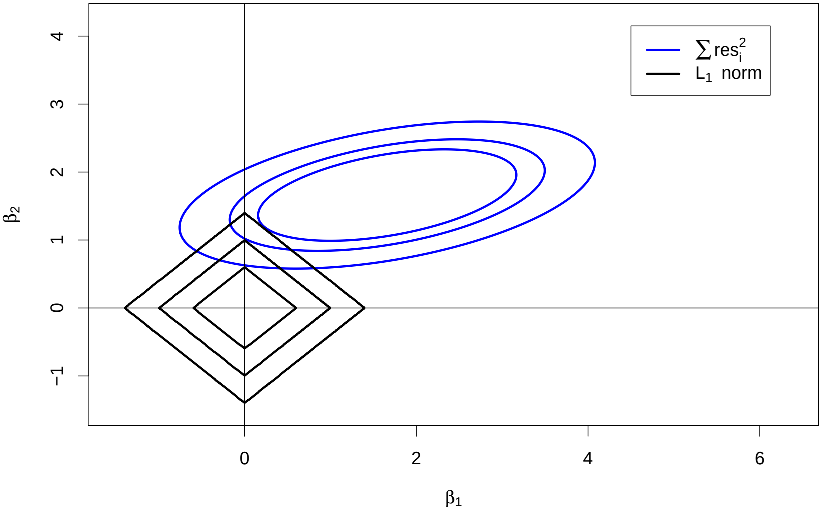


The regularised regression does a decent job: `err3.te`=113.4797069 (compare to `err1.te` and `err2.te` above).

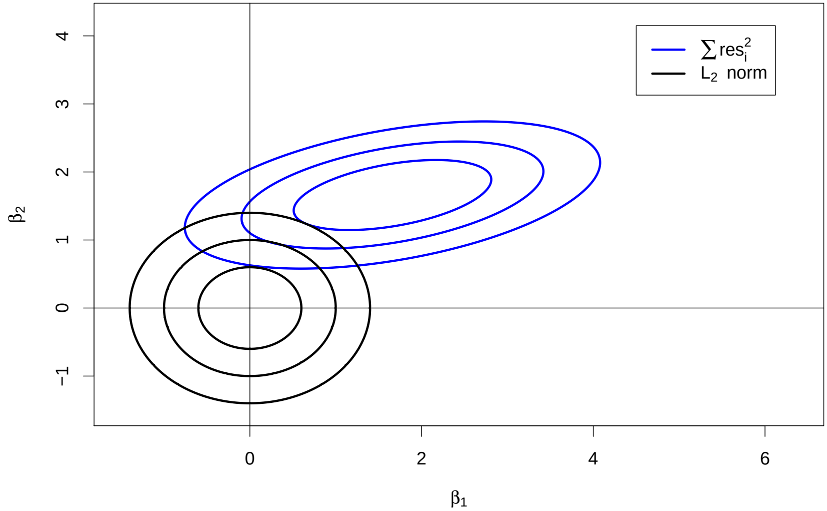
Lasso as a shrinkage method



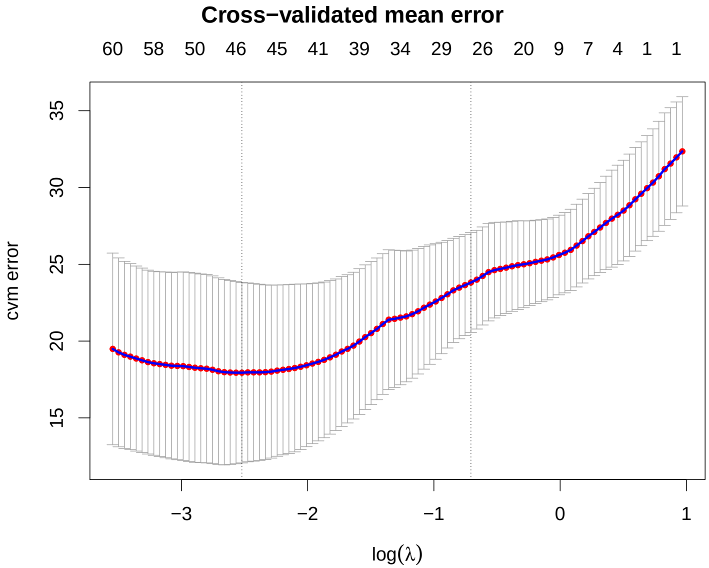
Lasso as a shrinkage method



Ridge regression - regularisation only



Cross-validation - Lasso



Linear regression in sklearn

```
import numpy as np
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
n_samples, n_features = 10, 5
np.random.seed(0)
y = np.random.randn(n_samples)
X = np.random.randn(n_samples, n_features)
lm = linear_model.LinearRegression()
lm.fit(X,y)
predictions = lm.predict(X)
print(predictions)
```

```
## [ 1.40977378  0.05327317  0.87351204  2.3979991   1.72087882 -0.6431204
##   0.88107802 -0.51570968  0.28830167  0.91424518]
```

```
print(lm.coef_)
```

```
## [ 1.64120907 -0.19746253 -1.1360001   0.04967949  2.13178637]
```

```
print(mean_squared_error(y, predictions))
```

```
## 0.0959236070718
```

```
print(r2_score(y, predictions))
```

```
## 0.89743447524
```

Ridge regression in sklearn

```
rlm = linear_model.Ridge(alpha=1.0)
rlm.fit(X, y)
predictions = rlm.predict(X)
print(mean_squared_error(y, predictions))
```

```
## 0.295837585119
```

```
print(r2_score(y, predictions))
```

```
## 0.683678105029
```

Lasso regression in sklearn

```
llm = linear_model.Lasso(alpha=0.1)
llm.fit(X,y)
predictions = llm.predict(X)
print(mean_squared_error(y, predictions))
```

```
## 0.444484380198
```

```
print(r2_score(y, predictions))
```

```
## 0.524738746861
```

Further topics

- Elastic net (ref. HTF 3.4)
- Least angle regression and its relation to LASSO (ref. HTF 3.4.4)
- http://scikit-learn.org/stable/modules/linear_model.html

Machine Learning for Finance

Classification

Dr Ning Wang

2019

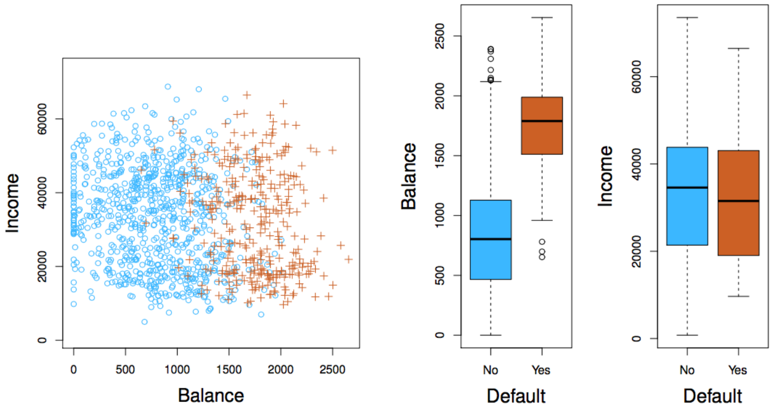
Classification problem

- Given $\{x_i\}_{i=1}^N$ *input*, *features*, *predictors*, *independent variables*
- Corresponding categorical $y_i \in \mathcal{C}$ *output*, *response*, *dependent variables*,
- Aim is to predict y given (out of sample) x, or
- Estimate $p(x) = \mathbb{P}(y = c|x)$ for $c \in \mathcal{C}$
    - binary classification: $|\mathcal{C}| = 2$
    - multiclass classification: $|\mathcal{C}| > 2$, *one vs. rest*, *one vs. one*

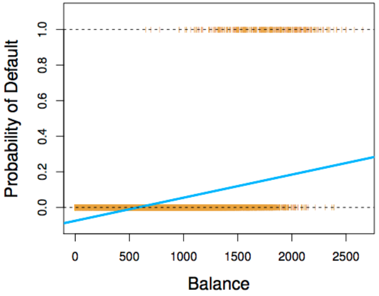
Some methods:

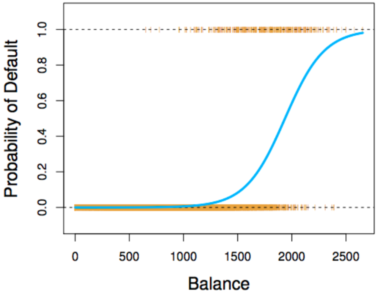
- Logistic regression
- Classification tree
- Random forests
- Support vector machines
- Neural networks
- etc.

Can we use linear regression?



Can we use linear regression?





- Linear regression does not estimate $\mathbb{P}(y = c|x)$ well.
- Linear regression $\mathbb{P}(y = c|x)$ can go to infinite.
- Linear regression is not appropriate multiple classification.

Binary logistic regression

In the binary classification case, let $\mathcal{C} = \{0, 1\}$

Given x find $p(x) = \mathbb{P}(y = 1|x)$ where $0 \leq p(x) \leq 1$

Binary classification:

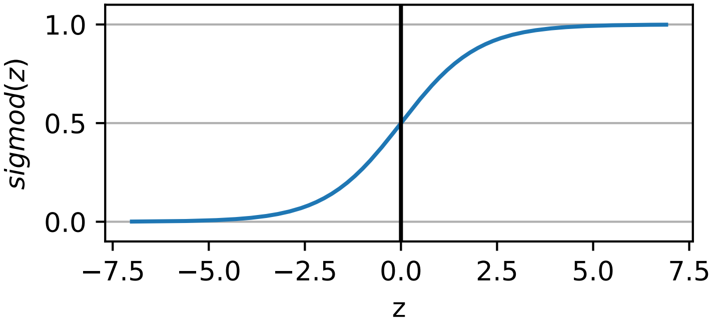
$$p(x) = sigmod(\beta_0 + \beta_1^T x)$$

$$sigmod(z) = \frac{1}{1 + e^{-z}}$$

$$p(x) = \frac{1}{1 + e^{\beta_0 + \beta_1^T x}}$$

Sigmod function

```
import matplotlib.pyplot as plt
import numpy as np
def sigmoid(z):
    return 1.0 / (1.0 + np.exp(-z))
z = np.arange(-7, 7, 0.1)
phi_z = sigmoid(z)
```



Multiclass logistic regression

$|\mathcal{C}| = m$

$$\mathbb{P}(y = c_i|x) = \frac{\exp(\beta_{i0}+\beta_i^T x)}{1+\sum_{j=1}^{m-1} \exp(\beta_{j0}+\beta_j^T x)}, \;\; i = 1, \ldots, m-1$$

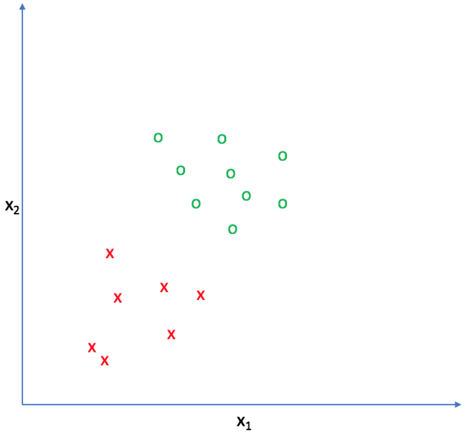
$$\mathbb{P}(y = c_m|x) = \frac{1}{1+\sum_{j=1}^{m-1} \exp(\beta_{j0}+\beta_j^T x)}$$

Decision boundary

Often the binary classification prediction is of the form:

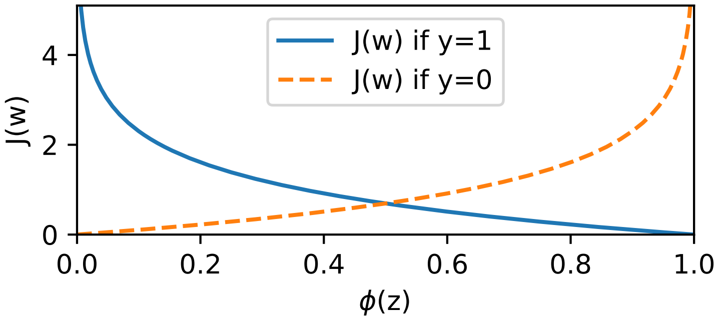
$$\hat{y} = p(y|x) = \begin{cases} 0 & \text{if } p(x) \leq k \\ 1 & \text{if } p(x) > k \end{cases}$$

for some function $p(x) \in \mathbb{R}$ and a well chosen threshold k.



Cost function

```
def cost_1(z):
    return - np.log(sigmoid(z))
def cost_0(z):
    return - np.log(1 - sigmoid(z))
z = np.arange(-10, 10, 0.1)
phi_z = sigmoid(z)
```



Cost function

Maximum likelihood is used to estimate the parameters:

$$\hat{\ell}(\beta) = \prod_{y=1} p(x_i;\beta) \prod_{y=0} (1 - p(x_i;\beta))$$

The log-likelihood can be written as

$$\begin{aligned}\ell(\beta) &= \sum_{i=1}^{N} y_i \log p(x_i;\beta) + (1 - y_i) \log(1 - p(x_i;\beta)) \\ &= \sum_{i=1}^{N} y_i(\beta_0 + \beta_1^T x_i) - \log(1 + exp(\beta_0 + \beta_1^T x_i))\end{aligned}$$

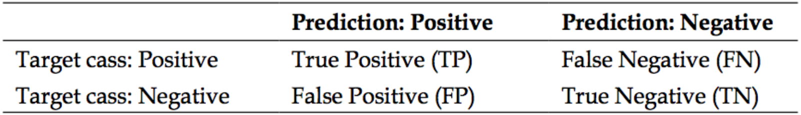
- The max of $\ell(\beta)$ can be approximated with Gradient descent, Conjugate gradient, Newton-Raphson algorithm. (HTF4.4.1)

Logistic regression

Logistic regression is implemented in the `sklearn.linear_model` package in `Python`.

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C=1000.0, random_state=0)
lr.fit(X_train, y_train)
```

Performance Evaluation - Confusion matrix



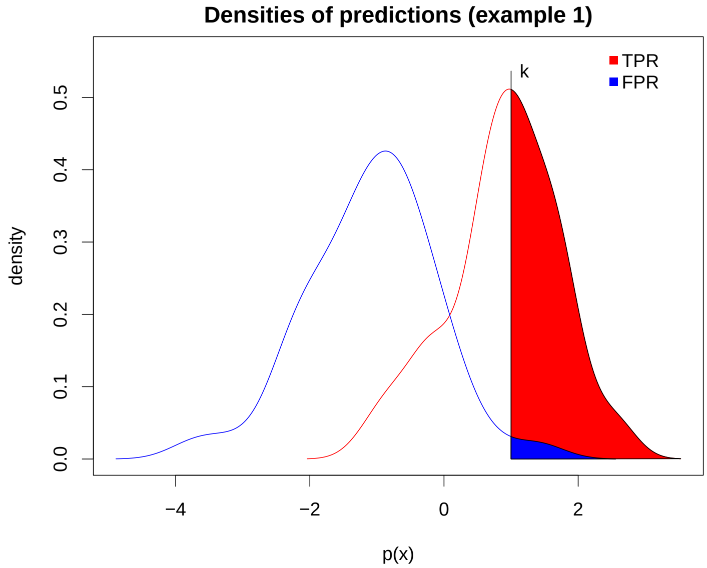
	Prediction: Positive	Prediction: Negative
Target cass: Positive	True Positive (TP)	False Negative (FN)
Target cass: Negative	False Positive (FP)	True Negative (TN)

- Accuracy: $|\mathrm{TP}+\mathrm{TN}|/|\mathrm{M}|$
- Precision: $|\mathrm{TP}|/|\mathrm{TP}+\mathrm{FP}|$
- Recall: $|\mathrm{TP}|/|\mathrm{TP}+\mathrm{FN}|$
- F1 score: $2 \times \mathrm{Precision} \times \mathrm{Recall}/|\mathrm{Precision} + \mathrm{Recall}$

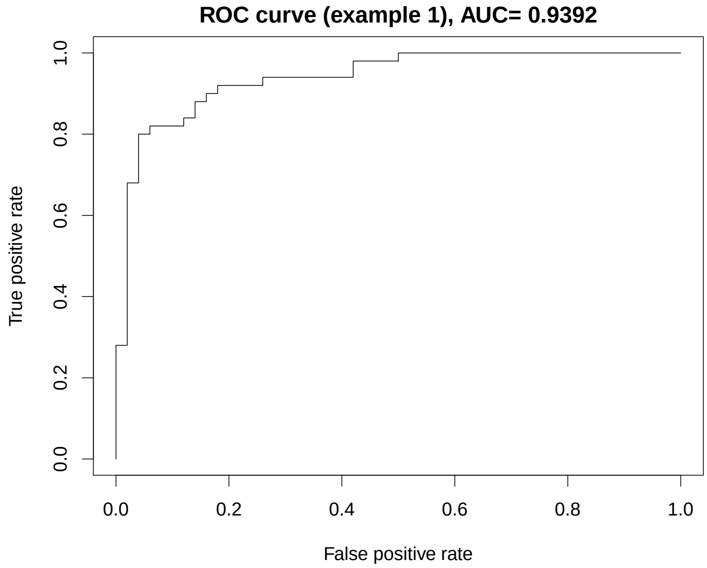
Receiver operating characteristic (ROC)

- True positive rate: $|\text{true positive}|/|\text{positive}|$
- False positive rate: $|\text{false positive}|/|\text{negative}|$
- Evaluates (and compares) performance of f on a given population $\{x_i\}_{i=1}^{N}$.
- Instead of fixing threshold k, ROC considers a whole range of k's and for each value of k plots the **false positive rate** against the **true positive rate**.
- **Area under ROC curve (AUC)** is used to summarize the overall performance. Higher AUC is good.

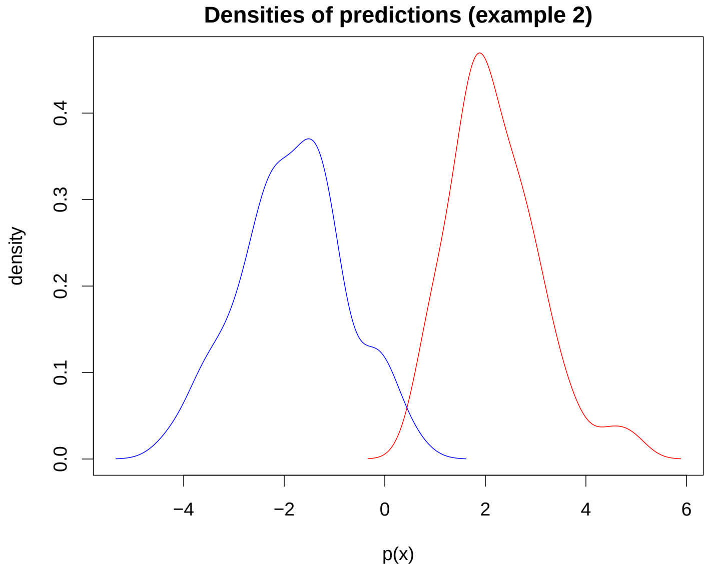
Receiver operating characteristic (ROC)



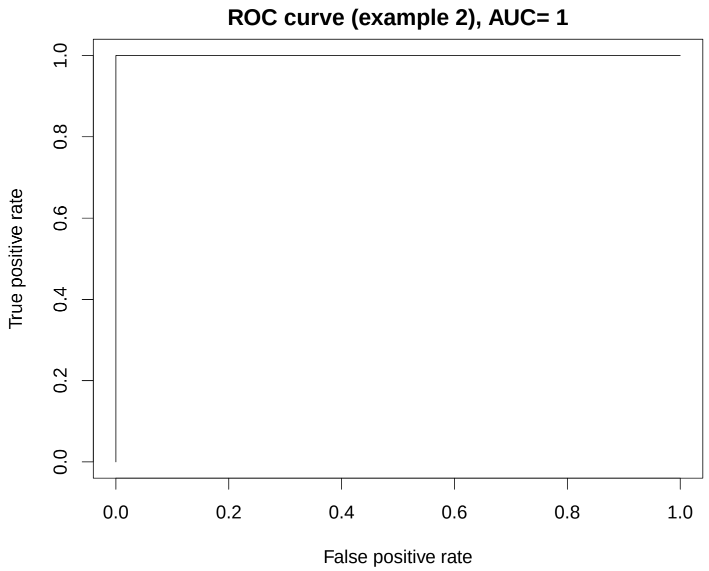
Receiver operating characteristic (ROC)



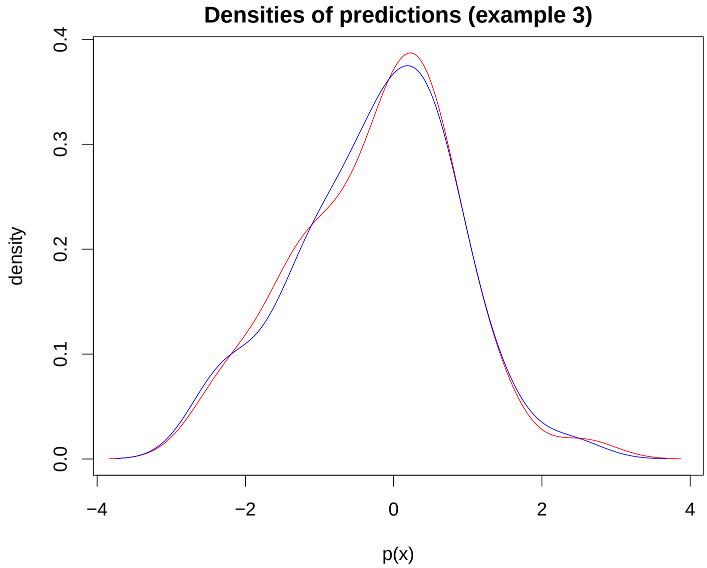
Receiver operating characteristic (ROC)



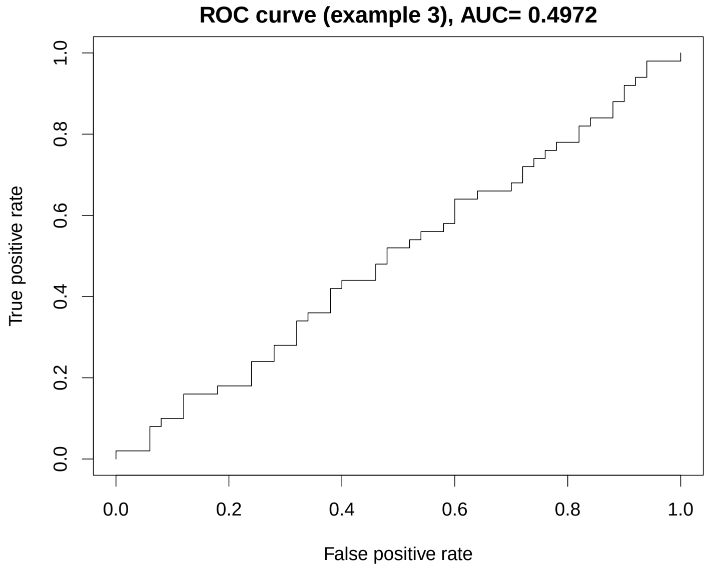
Receiver operating characteristic (ROC)



Receiver operating characteristic (ROC)



Receiver operating characteristic (ROC)



ROC in Python

```
from sklearn import svm, datasets
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
plt.clf()
breast_cancer = load_breast_cancer()
print(breast_cancer.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']
```

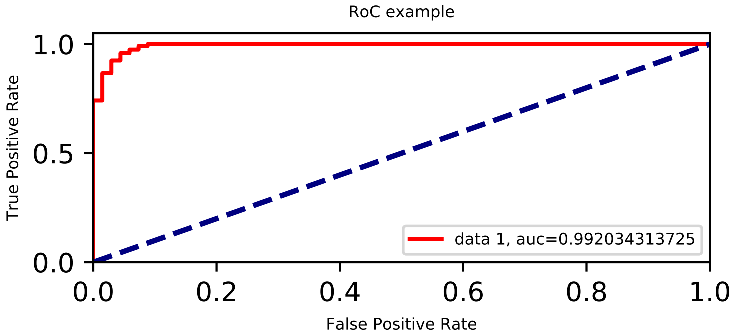
ROC in Python

```
X = breast_cancer.data
y = breast_cancer.target
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.33, random_state=44)
#sklearn.linear_model.LogisticRegression(penalty='l2', dual=False, tol=0.0001,
# C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None,
#solver='warn', max_iter=100, multi_class='warn', verbose=0, warm_start=False, n_jobs=None)
clf = LogisticRegression(penalty='l2', C=0.1)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("Accuracy", metrics.accuracy_score(y_test, y_pred))
```

```
## ('Accuracy', 0.9521276595744681)
```

ROC in Python

```
y_pred_proba = clf.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc), color='red')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate',fontsize=6)
plt.ylabel('True Positive Rate',fontsize=6)
plt.title('RoC example',fontsize=6)
plt.legend(loc="lower right",fontsize=6)
plt.show()
```



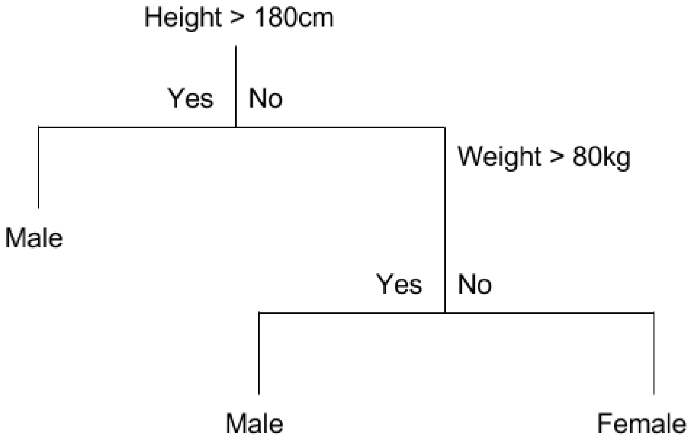
Classification tree

- Classification trees use splitting rules to segment the predictor into regions $R_1, \ldots, R_M$.
- We estimate

$$\mathbb{P}(y = c_i | x \in R_j) \approx \frac{1}{N_j} \sum_{x_i \in R_j} \mathbf{1}(y_i = c_i) =: p_{ji}$$

where $N_j = \#\{x_i \in R_j\}$.

Classification tree



Error functions

- Predicted class: $i(j) = \text{argmax}_i \, p_{ji}$.
- Measures of error $Q(R_j) = Q(j)$ in R_j:
  - Miss-classification error: $1 - p_{ji(j)}$,
  - Gini index: $\sum_{k=1}^{|\mathcal{C}|} p_{ji}(1 - p_{ji})$
  - Cross-entropy: $-\sum_{k=1}^{|\mathcal{C}|} p_{ji} \log p_{ji}$

Classification tree - the algorithm

The exact optimum is expensive to compute. Good approximations can be find.

Growing the tree

- The decision tree is determined by recursive partitioning (splitting (sub)regions in two).
- For a given region R, find j and s, such that

$$R_l(j,s) := \{x_i | x_i^j \leq s\},\ R_u(j,s) := \{x_i | x_i^j > s\},$$

  and j, s minimises:

$$|R_l(j,s)| Q(R_l(j,s)) + |R_u(j,s)| Q(R_u(j,s)).$$

- Grow the tree until a minimum node size is reached.

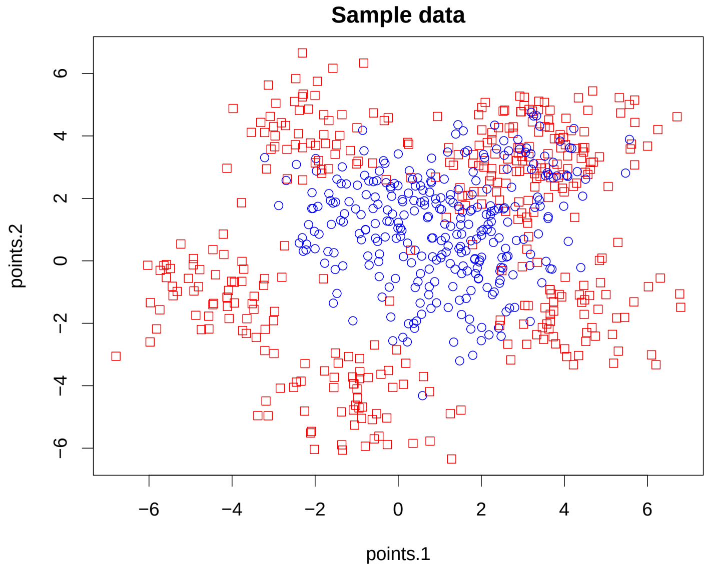
Classification tree - the algorithm

Pruning the tree

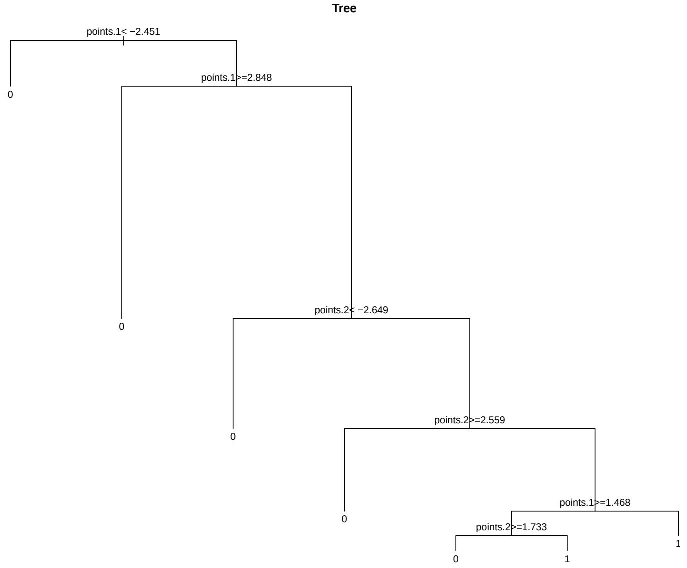
- Given a large (over-fitted) tree T_0 defined by the regions $R_1, \ldots, R_{|T_0|}$, find a sub-tree T_α that minimises the *cost complexity criterion*:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q(R_m) + \alpha |T|.$$

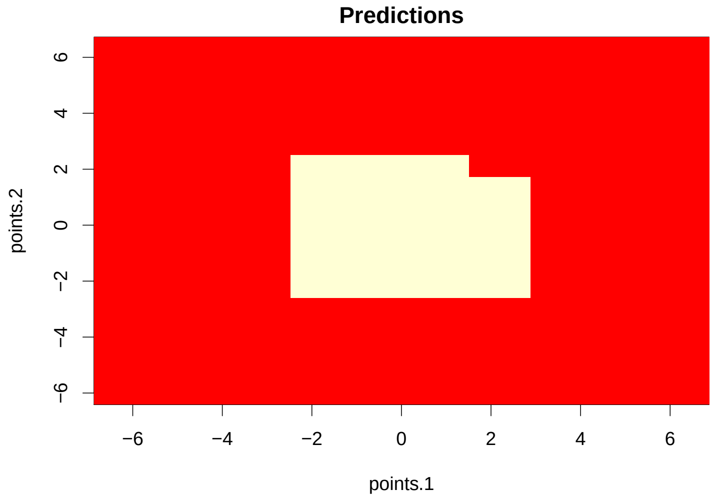
Classification tree - example



Classification tree - example - training

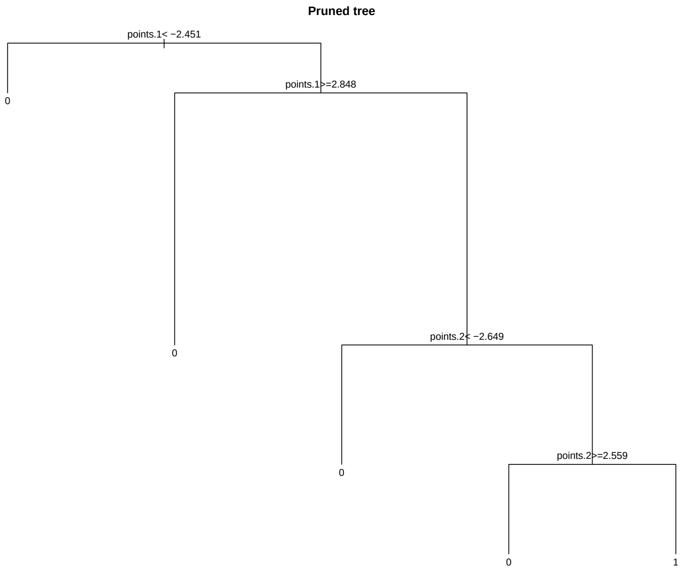


Classification tree - fit on full tree

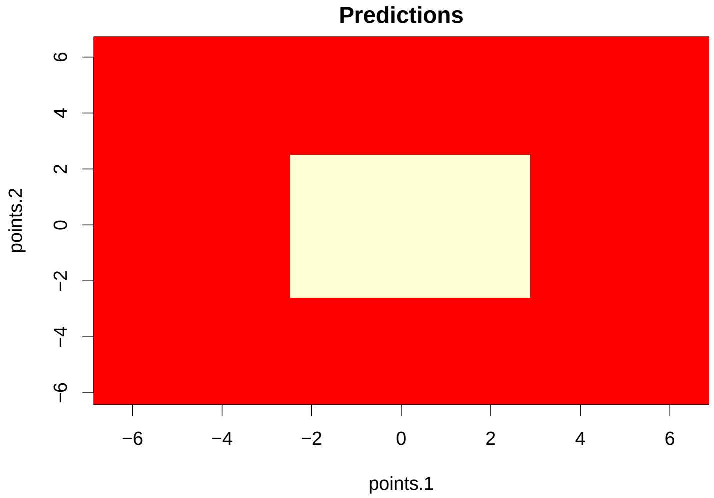


Accuracy on `test set` is 0.8245614 compared to 0.8576998 on `training set`.

Classification tree - example - pruning



Classification tree - fit on pruned tree



Accuracy on `test set` is 0.8187135 compared to 0.8479532 on `training set`.

Classification Tree - Pros and Cons

- Tree methods are simple and useful for interpretation.
- Performance is not competitive with the best supervised learning approaches.
- Bagging and boosting are used to grow multiple trees which are combined to yield a single consensus prediction.
- Combining a large number of trees can often result in dramatic improvements in prediction accuracy.

Random Forests

Let the (weak) predictor be classification/regression tree T. The *aim* is to build a large number of *de-correlated* trees and bag them.

The algorithm:

1. For $m = 1, \dots M$
   - draw a bootstrap sample $\mathcal{S}_m$ from $\mathcal{S}$ of a predefined size.
   - grow a tree T_m to $\mathcal{S}_m$ until a predefined minimum node size is reached by repeating the following steps:
     - select k factors/dimension at random out of d $(X \subseteq \mathbb{R}^d)$,
     - pick the best split point among the k,
     - split the node into two child nodes.
2. Define the predictor f^*_{fr} by bagging $\{T_m\}_{m=1}^M$.

Random Forests

Properties:

- random forests tend to perform well on trees (that may be noisy but have small bias),
- performance depends on the choice of k,
- best practice *default* value for k: $\lfloor \sqrt{d} \rfloor$ for classification,
- best practice *default* value for k: $\lfloor d/3 \rfloor$ for regression.

Python Examples - data preparation

```
from sklearn.cross_validation import train_test_split
from sklearn import datasets
iris = datasets.load_iris()
print(iris.feature_names)
```

```
## ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
```

```
print(iris.target_names)
```

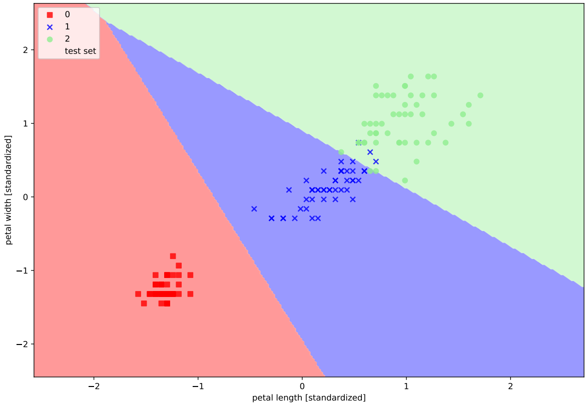
```
## ['setosa' 'versicolor' 'virginica']
```

```
X = iris.data[:, [2, 3]]
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(
         X, y, test_size=0.3, random_state=0)
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
```

Python Examples - Logistic Regression

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(C=1000.0, random_state=0)
lr.fit(X_train_std, y_train)
y_pred = lr.predict(X_test)
print("Accuracy", metrics.accuracy_score(y_test, y_pred))
```

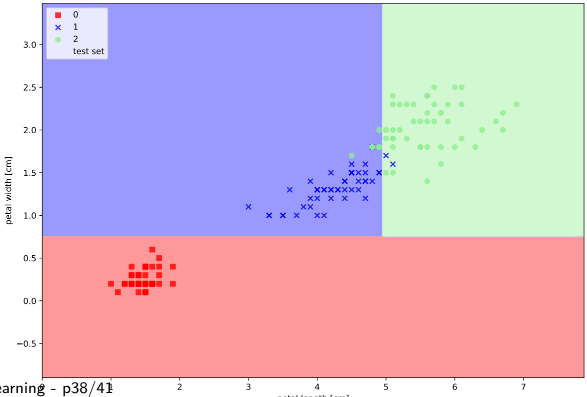
```
## ('Accuracy', 0.24444444444444444)
```



Python Examples - Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
# sklearn.tree.DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None,
# min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None,
# random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None,
# class_weight=None, presort=False)
tree = DecisionTreeClassifier(criterion='entropy', max_depth=2, random_state=1)
tree.fit(X_train, y_train)
y_pred = tree.predict(X_test)
print("Accuracy", metrics.accuracy_score(y_test, y_pred))
```

```
## ('Accuracy', 0.91111111111111109)
```



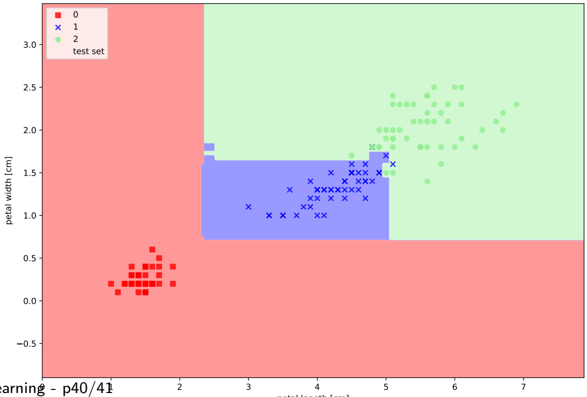
Python Examples - Pruning tree

- Pruning helps us to avoid overfitting
- Any additional split that does not add significant value is not worth while.
- Avoid overfitting by changing the parameters like `max_leaf_nodes`, `min_samples_leaf`, `max_depth`
  - `max_leaf_nodes`: Reduce the number of leaf nodes
  - `min_samples_leaf`: Restrict the size of sample leaf
  - `max_depth`: Reduce the depth of the tree to build a generalized tree

Python Examples - Random Forest

```
from sklearn.ensemble import RandomForestClassifier
# sklearn.ensemble.RandomForestClassifier(n_estimators='warn', criterion='gini', max_depth=None,
# min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto',
# max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True,
# oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None)
forest = RandomForestClassifier(criterion='entropy',n_estimators=50, random_state=1)
forest.fit(X_train, y_train)
y_pred = forest.predict(X_test)
print("Accuracy", metrics.accuracy_score(y_test, y_pred))
```

```
## ('Accuracy', 0.97777777777777775)
```



Further topics

- k-nearest neighbour classifier (13.3 HTF)

Machine Learning for Finance

Support Vector Machines

Dr Ning Wang

2019

This lecture

- Support vector machines, linear & separable case (3.1-2 Burgess)
- Support vector machines, linear & non-separable case (3.5 Burgess)
- Support vector machines, non-linear case (4 Burgess)
- Examples

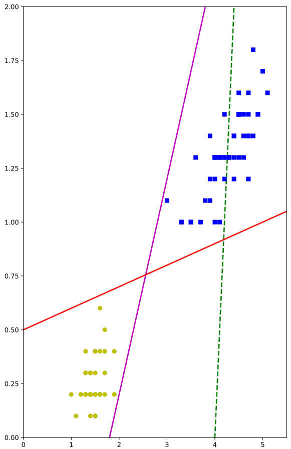
Main literature:

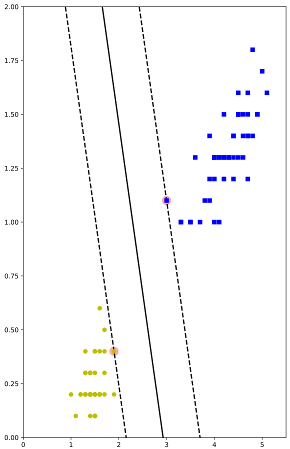
- *A Tutorial on Support Vector Machines for Pattern Recognition* by C. Burgess, Data mining and knowledge discovery 2.2 (1998): 121-167.

Further read:

- *The Elements of Statistical Learning* by T. Hastie, R. Tibshirani and J. Friedman, Second edition, Springer, 2009

Support vector machine, separable case





Support vector machine, separable case

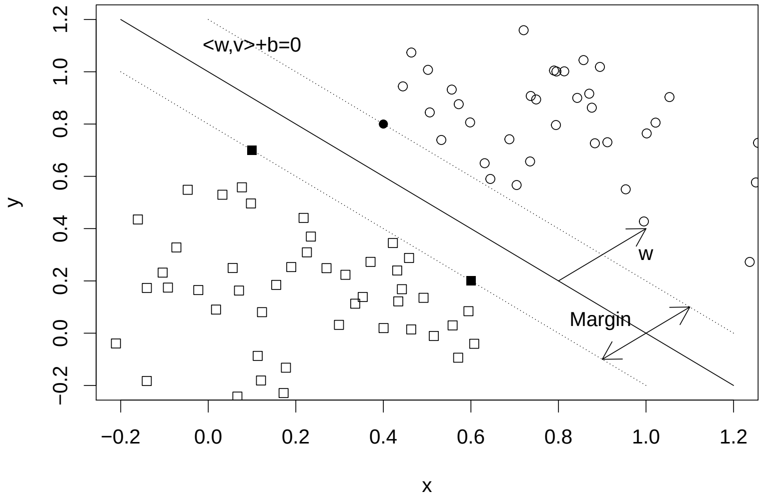
Observations: points $v_i \in \mathbb{R}^d$ and labels $u_i \in \{-1, 1\}$ for $i = 1, \ldots, N$.

Linear decision function:

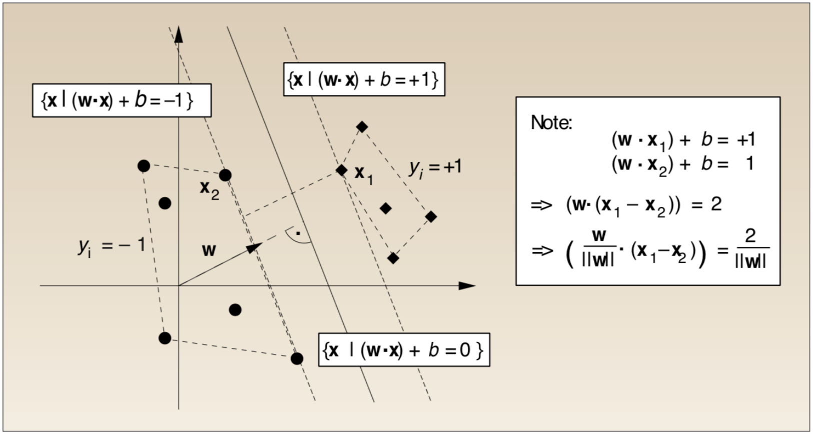
$$\begin{aligned} \langle w, v_i \rangle + b \geq 0, &\text{ for } u_i = 1 \\ \langle w, v_i \rangle + b \leq 0, &\text{ for } u_i = -1 \end{aligned}$$

Several w, b suitable pairs may exist. Which one is the “best”?

Support vector machine, separable case



Support vector machine - a bit of geometry



$\{\mathbf{x} \mid (\mathbf{w}\cdot\mathbf{x}) + b = -1\}$
$\{\mathbf{x} \mid (\mathbf{w}\cdot\mathbf{x}) + b = +1\}$
$\{\mathbf{x} \mid (\mathbf{w}\cdot\mathbf{x}) + b = 0\}$
$\mathbf{x}_2$
$\mathbf{x}_1$
$y_i = +1$
$y_i = -1$
$\mathbf{w}$
Note:
$(\mathbf{w}\cdot\mathbf{x}_1) + b = +1$
$(\mathbf{w}\cdot\mathbf{x}_2) + b = 1$
$\Rightarrow (\mathbf{w}\cdot(\mathbf{x}_1 - \mathbf{x}_2)) = 2$
$\Rightarrow \left(\frac{\mathbf{w}}{\|\mathbf{w}\|}\cdot(\mathbf{x}_1 - \mathbf{x}_2)\right) = \frac{2}{\|\mathbf{w}\|}$

HEARST, Marti A., et al., 1998. Support vector machines. IEEE Intelligent Systems, 13(4), 18-28.

Support vector machine, separable case

Formulation as an optimisation problem:

The aim is to find the separating hyper-plane with the largest margin:

$$\underset{w,b}{\operatorname{argmax}} \frac{2}{\|w\|} = \underset{w,b}{\operatorname{argmin}} \frac{1}{2}\|w\|^2.$$

constrained to:

$$\langle w, v_i \rangle + b \geq 1, \text{ for } u_i = 1$$
$$\langle w, v_i \rangle + b \leq -1, \text{ for } u_i = -1$$

or in short:

$$u_i(\langle w, v_i \rangle + b) - 1 \geq 0, \text{ for } i = 0, \ldots, N.$$

The penalisation factor C is to be tuned by cross-validation.

Support vector machine, separable case

Lagrangian formulation

$$L_P = \tfrac{1}{2}\|w\|^2 - \sum_{i=1}^{N} \alpha_i u_i(\langle w, v_i\rangle + b) + \sum_{i=1}^{N} \alpha_i .$$

required: $\alpha_i \geq 0$ for $i = 1, \ldots, N$.

Karush-Kuhn-Tucker conditions (necessary and sufficient for convex problems) for $i = 1, \ldots, N$:

$$\begin{aligned}
\tfrac{\partial}{\partial w} L_P = w - \sum_{i=1}^{N} \alpha_i u_i v_i &= 0,\\
\tfrac{\partial}{\partial b} L_P = -\sum_{i=1}^{N} \alpha_i u_i &= 0,\\
u_i(\langle w, v_i\rangle + b) - 1 &\geq 0,\\
\alpha &\geq 0,\\
\alpha_i(u_i(\langle w, v_i\rangle + b) - 1) &= 0.
\end{aligned}$$

Support vector machine, separable case

The condition $\alpha_i(u_i(\langle w, v_i\rangle + b) - 1) = 0$ implies that at least one of

- $\alpha_i = 0$,
- $u_i(\langle w, v_i\rangle + b) = 1$

must hold.

If $\alpha_i > 0$, that is when v_i is on the margin, then v_i is a **support vector**, hence the name.

Support vector machine, separable case

From KKT, we have:

$$w = \sum_{i=1}^{N} \alpha_i u_i v_i, \qquad (*)$$

$$0 = \sum_{i=1}^{N} \alpha_i u_i. \qquad (**)$$

Dual problem

Maximise:

$$L_D = \sum_{i=1}^{N} \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j u_i u_j \langle v_i, v_j \rangle.$$

subject to constraints (KKT).

Support vector machine, non-separable case

Formulation as an optimisation problem:

In the non-separable case we allow points within the margin, but at a cost/penalty:

$$\underset{w,b}{\operatorname{argmin}} \tfrac{1}{2}\|w\|^2 + C\left(\sum_{i=1}^{N} \xi_i\right).$$

constrained to:

$$\begin{gathered} \langle w, v_i \rangle + b \geq 1 - \xi_i, \text{ for } u_i = 1 \\ \langle w, v_i \rangle + b \leq -1 + \xi_i, \text{ for } u_i = -1 \\ \xi_i \geq 0, \quad i = 1, \ldots, N \end{gathered}$$

Support vector machine, non-separable case

Lagrangian formulation:

$$L_P = \tfrac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i\{u_i(\langle w, v_i\rangle + b) - 1 + \xi_i\} - \sum_{i=1}^{N}\mu_i\xi_i.$$

The KKT conditions: for $i = 1, \ldots, N$ (*), (**), and:

$$\begin{aligned}
\tfrac{\partial}{\partial \xi_i} L_P = C - \alpha_i - \mu_i &= 0,\\
u_i(\langle w, v_i\rangle + b) - 1 + \xi_i &\geq 0,\\
\xi_i &\geq 0,\\
\alpha_i &\geq 0,\\
\mu_i &\geq 0,\\
\alpha_i\left\{u_i(\langle w, v_i\rangle + b) - 1 + \xi_i\right\} &= 0,\\
\mu_i\xi_i &= 0.
\end{aligned}$$

Support vector machine, non-separable case

The condition $\alpha_i \{u_i(\langle w, v_i \rangle + b) - 1 + \xi_i\} = 0$ implies that at least one of

- $\alpha_i = 0$,
- $u_i(\langle w, v_i \rangle + b) = 1 - \xi_i$

must hold.

If $\alpha_i > 0$, that is when v_i is on the margin or on the wrong side of the margin, then v_i is a **support vector**.

Support vector machine, non-separable case

Dual formulation:

Maximise:

$$L_D = \sum_{i=1}^{N} \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j u_i u_j \langle v_i, v_j \rangle$$

subject to

$$0 \leq \alpha_i \leq C,$$

$$\sum_{i=1}^{N} \alpha_i u_i = 0.$$

Non-linear support vector machines

If a hyper-plane does not work on the original data in the original space, it may work on the transformed data in another space.

That is, for some function $\Phi : \mathbb{R}^d \to \mathcal{H}$, a hyper-plane might be a good separator, that maximises

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j u_i u_j \langle \Phi(v_i), \Phi(v_j) \rangle$$

subject to

$$0 \leq \alpha_i \leq C,$$

$$\sum_{i=1}^{N} \alpha_i u_i = 0.$$

Non-linear support vector machines

Note that the optimisation depends on the v_i's only through:

$$\langle \Phi(v_i), \Phi(v_j) \rangle.$$

Generalisation: replace the inner product with kernel

$$K(v_i, v_j).$$

Mercer's condition: Given $K(,)$, there exists a pair $\{\mathcal{H}, \Phi\}$ such that $K(x, y) = \langle \Phi(x), \Phi(y) \rangle$ if and only if

$$\int g(x)^2 dx < \infty$$

implies

$$\int K(x, y) g(x) g(y) dx dy \geq 0.$$

Non-linear support vector machines

Examples of kernel functions:

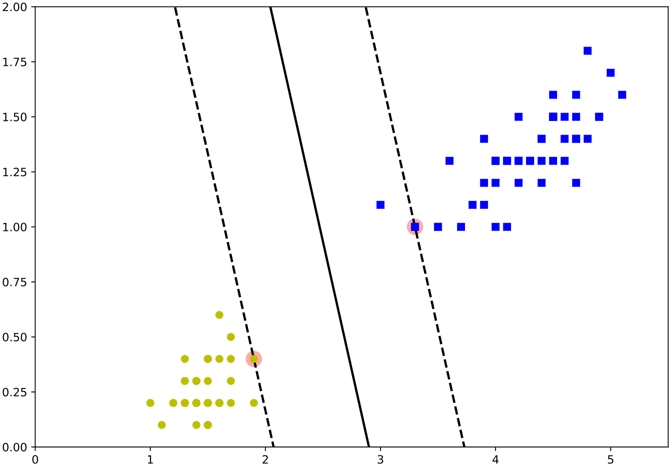
$$K(x,y) = \langle x, y\rangle^p, \text{ polynomial}$$
$$K(x,y) = e^{-\gamma\|x-y\|^2}, \text{ radial basis}$$
$$K(x,y) = \tanh(\kappa\langle x, y\rangle - \delta), \text{ sigmoid}$$

SVM Classifier in sklearn

```
# SVM Classifier
from sklearn.svm import SVC
from sklearn import datasets
from sklearn import metrics
from sklearn.model_selection import train_test_split
iris = datasets.load_iris()
X = iris["data"][:, (2, 3)]  # petal length, petal width
y = iris["target"]
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3)
# sklearn.svm.SVC(C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0,
# shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None,
# verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)
svm_clf = SVC(kernel="linear", C= 5)
svm_clf.fit(X_train, y_train)
y_pred = svm_clf.predict(X_test)
print("Accuracy", metrics.accuracy_score(y_test, y_pred))
```

```
## ('Accuracy', 1.0)
```

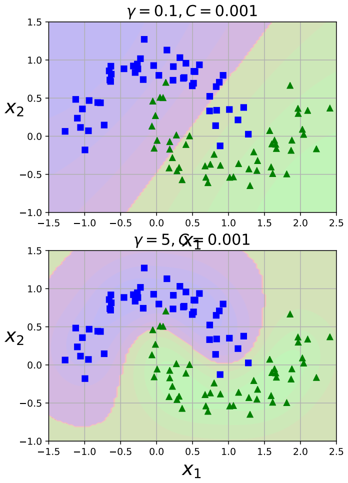
SVM Classifier in sklearn



SVM Classifier in sklearn

```
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=100, noise=0.15, random_state=42)
gamma1, gamma2 = 0.1, 5
C1, C2 = 0.001, 1000
hyperparams = ((gamma1, C1), (gamma1, C2),
               (gamma2, C1), (gamma2, C2))
svm_clfs = []
for gamma, C in hyperparams:
    rbf_kernel_svm_clf =SVC(kernel="rbf", gamma=gamma, C=C)
    rbf_kernel_svm_clf.fit(X, y)
    svm_clfs.append(rbf_kernel_svm_clf)
```

SVM Classifier in sklearn



Further topics

- Numerical implementation of SVM for large data-sets and/or high dimensions (ref. 5 Burgess, 12.3.5 HFT)
- SVMs for regression (ref. 12.3.6 HTF)

Machine Learning for Finance

Neural Networks

Dr Ning Wang

2019

Artificial Intelligence

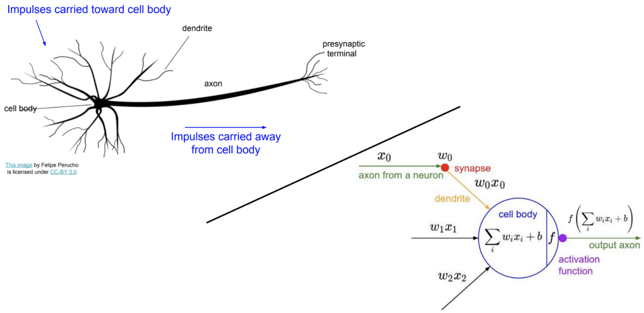
- The idea of AI is to teaching machine to behave more like the human brain.
- Neural Networks move machine learning closer to AI.
  - Make learning algorithms much better and easier to use.
  - Make revolutionary advances in machine learning and neuroscience.

Single Neuron

In 1943, neurophysiologist Warren McCulloch and mathematician Walter Pitts wrote a paper on how neurons might work. In order to describe how neurons in the brain might work, they modeled a simple neural network using electrical circuits [1].

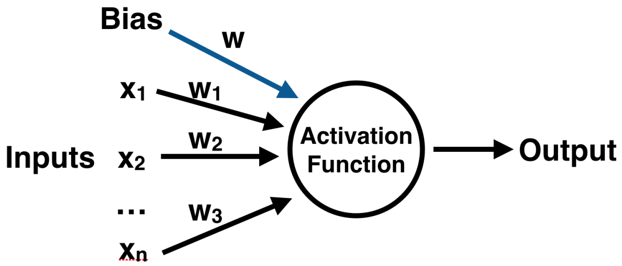
[1] http://www.cs.cmu.edu/~./epxing/Class/10715/reading/McCulloch.and.Pitts.pdf

Single Neuron



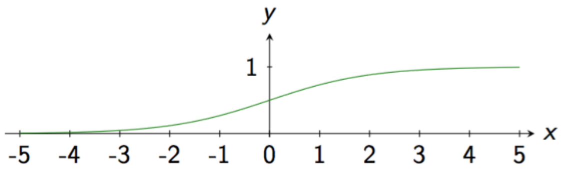
- Input layer: An input $X = (x1, ..., xn)$
- Activation: weighted sum of input features
- Activation function: logistic function $f(.)$ applied to the weighted sum

Multiple Input Neuron

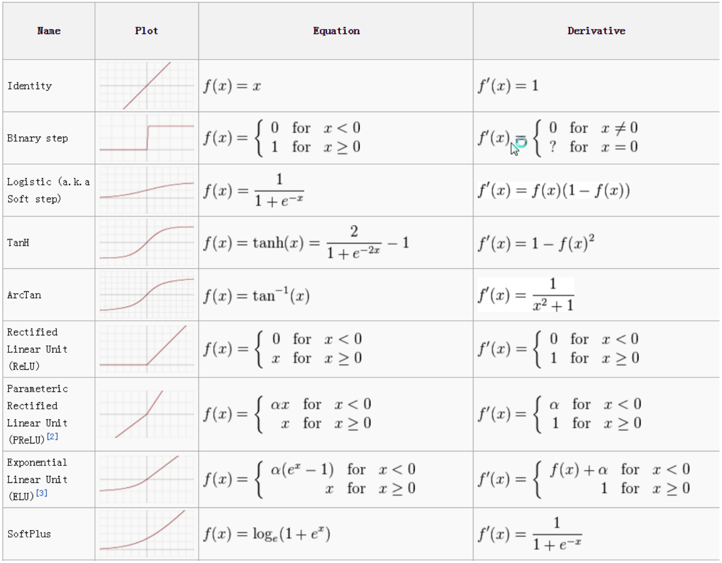


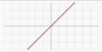
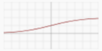
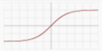
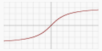
Activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$



Activation functions [2]



Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

[2] https://towardsdatascience.com

Multilayer Perceptron (MLP)

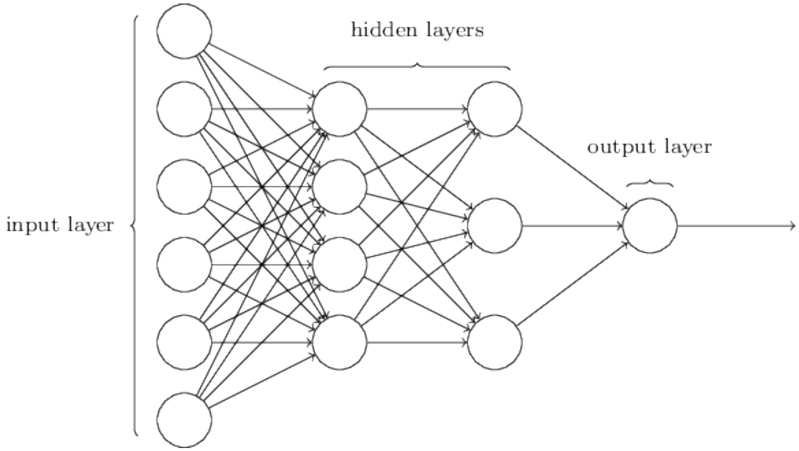
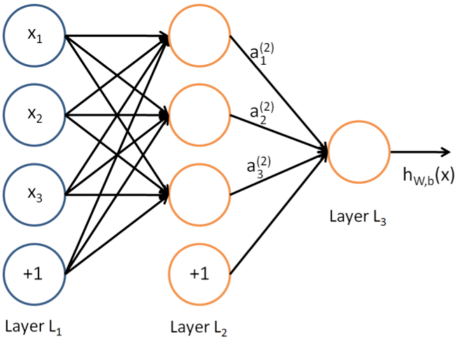


Image Souce: neuralnetworksanddeeplearning.com

Multilayer Perceptron (MLP)

- Feed-forward: MLP maps sets of input data onto a set of appropriate outputs.
- Fully-connected: MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one.
- Nonlinear: Except for the input nodes, each node is a neuronwith a nonlinear activation function.
- Backpropagation: MLP utilizes backpropagation for training the network.

ForwardFeeding



$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$
$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$
$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$
$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

Backpropagation (BP)

- BP is a common method of training artificial neural networks.
- BP propagates the errors backward and adjust the weights.
- BP is normally used with an optimization method such as gradient descent.
- BP calculates the gradient of a loss function with respect to all the weights in the network.
- The gradient is fed to the optimization method which in turn uses it to update the weights, in an attempt to minimize the loss function.

Back Propagation

$$J(W,b;x,y) = \frac{1}{2}\left\| h_{W,b}(x) - y \right\|^2$$

$$\begin{aligned} J(W,b) &= \left[\frac{1}{m} \sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)}) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2 \\ &= \left[\frac{1}{m} \sum_{i=1}^{m} \left(\frac{1}{2} \left\| h_{W,b}(x^{(i)}) - y^{(i)} \right\|^2 \right) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{(l)} \right)^2 \end{aligned}$$

Back Propagation

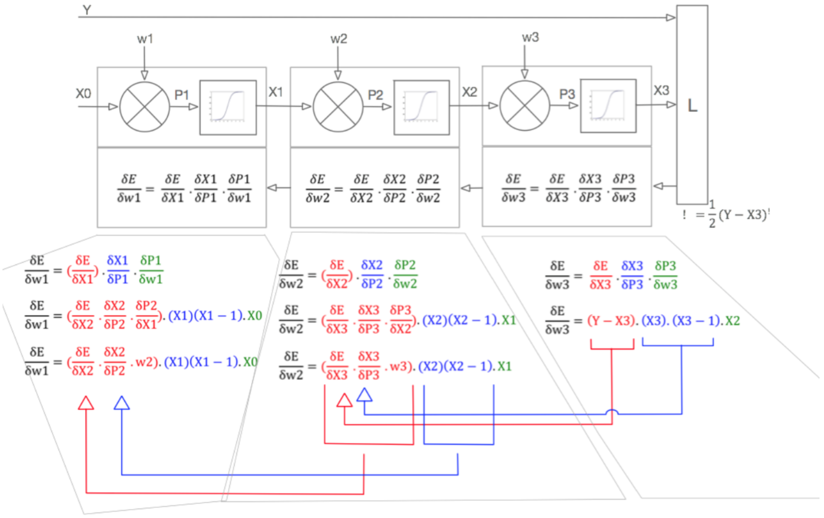
$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b)$$

$$b_{i}^{(l)} = b_{i}^{(l)} - \alpha \frac{\partial}{\partial b_{i}^{(l)}} J(W,b)$$

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x^{(i)},y^{(i)}) \right] + \lambda W_{ij}^{(l)}$$

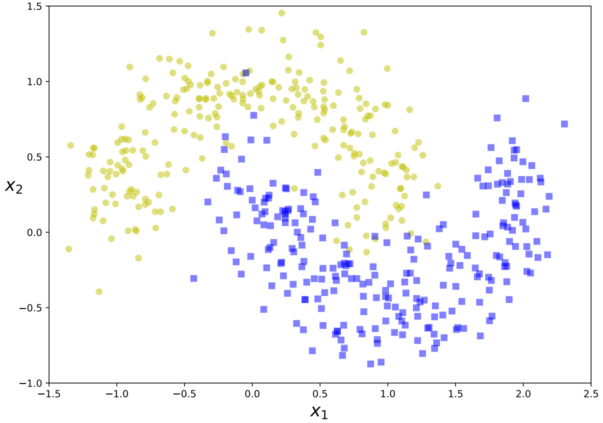
$$\frac{\partial}{\partial b_{i}^{(l)}} J(W,b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b_{i}^{(l)}} J(W,b;x^{(i)},y^{(i)})$$

Back Propagation


Y
w1
w2
w3
X0
P1
X1
P2
X2
P3
X3
L
$\frac{\delta E}{\delta w1} = \frac{\delta E}{\delta X1} \cdot \frac{\delta X1}{\delta P1} \cdot \frac{\delta P1}{\delta w1}$
$\frac{\delta E}{\delta w2} = \frac{\delta E}{\delta X2} \cdot \frac{\delta X2}{\delta P2} \cdot \frac{\delta P2}{\delta w2}$
$\frac{\delta E}{\delta w3} = \frac{\delta E}{\delta X3} \cdot \frac{\delta X3}{\delta P3} \cdot \frac{\delta P3}{\delta w3}$
$! = \frac{1}{2}(Y - X3)^{!}$
$\frac{\delta E}{\delta w1} = (\frac{\delta E}{\delta X1}) \cdot \frac{\delta X1}{\delta P1} \cdot \frac{\delta P1}{\delta w1}$
$\frac{\delta E}{\delta w1} = (\frac{\delta E}{\delta X2} \cdot \frac{\delta X2}{\delta P2} \cdot \frac{\delta P2}{\delta X1}).(X1)(X1 - 1).X0$
$\frac{\delta E}{\delta w1} = (\frac{\delta E}{\delta X2} \cdot \frac{\delta X2}{\delta P2} .w2).(X1)(X1 - 1).X0$
$\frac{\delta E}{\delta w2} = (\frac{\delta E}{\delta X2}) \cdot \frac{\delta X2}{\delta P2} \cdot \frac{\delta P2}{\delta w2}$
$\frac{\delta E}{\delta w2} = (\frac{\delta E}{\delta X3} \cdot \frac{\delta X3}{\delta P3} \cdot \frac{\delta P3}{\delta X2}).(X2)(X2 - 1).X1$
$\frac{\delta E}{\delta w2} = (\frac{\delta E}{\delta X3} \cdot \frac{\delta X3}{\delta P3} .w3).(X2)(X2 - 1).X1$
$\frac{\delta E}{\delta w3} = \frac{\delta E}{\delta X3} \cdot \frac{\delta X3}{\delta P3} \cdot \frac{\delta P3}{\delta w3}$
$\frac{\delta E}{\delta w3} = (Y - X3).(X3).(X3 - 1).X2$

MLP example - generating data

```
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
X, y = make_moons(n_samples=500, noise=0.2, random_state=18)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

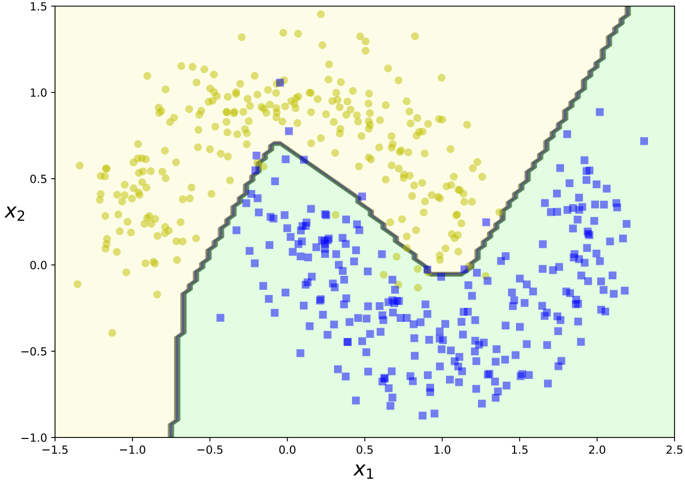


MLPClassifier in sklearn

```
from sklearn.neural_network import MLPClassifier
#MLPClassifier(hidden_layer_sizes=(100, ), activation='relu', solver='adam', alpha=0.0001, batch_size='auto
#learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True, random_state=
#tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False,
#validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10)
mlp = MLPClassifier(hidden_layer_sizes=(10, 10, 10), max_iter=1000)
mlp.fit(X_train, y_train)
predictions = mlp.predict(X_test)
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report, confusion_matrix
print(accuracy_score(y_test,predictions))
```

```
## 0.952
```

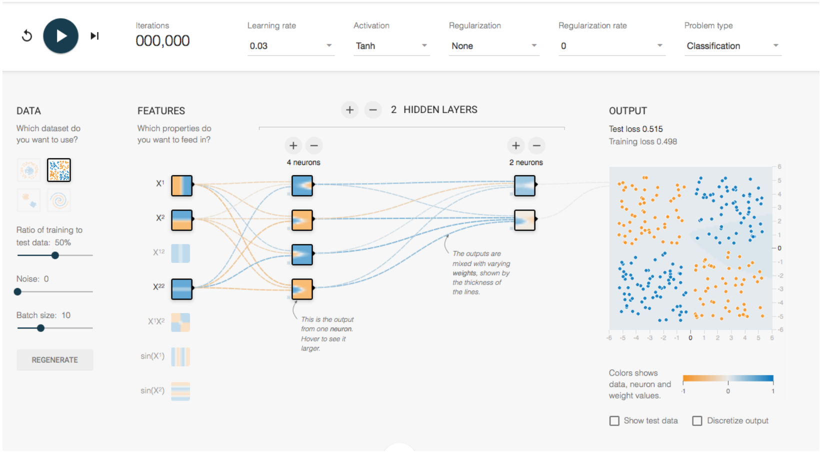
MLPClassifier in sklearn



Tips on MLPClassifier

- Multi-layer Perceptron is sensitive to feature scaling, so it is highly recommended to scale your data.
- Sale each attribute on the input vector X to [0, 1] or [-1, +1], or standardize it to have mean 0 and variance 1.
- Finding a reasonable regularization parameter using GridSearchCV
- L-BFGS converges faster and with better solutions on small datasets. For relatively large datasets, however, Adam is very robust. It usually converges quickly and gives pretty good performance. SGD with momentum or nesterov's momentum, on the other hand, can perform better than those two algorithms if learning rate is correctly tuned.

Visualizing Neural Nets



Iterations
000,000

Learning rate: 0.03

Activation: Tanh

Regularization: None

Regularization rate: 0

Problem type: Classification

DATA

Which dataset do you want to use?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

FEATURES

Which properties do you want to feed in?

2 HIDDEN LAYERS

4 neurons

2 neurons

This is the output from one neuron. Hover to see it larger.

The outputs are mixed with varying weights, shown by the thickness of the lines.

OUTPUT

Test loss 0.515
Training loss 0.498

Colors shows data, neuron and weight values.

☐ Show test data ☐ Discretize output

http://playground.tensorflow.org is a website where you can tweak and visualize neural networks.

TensorFlow

- TensorFlow is an open source software library for numerical computation using data flow graphs.
- TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence.
- The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API.

Keras

- Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.
- Keras is developed with a focus on enabling fast experimentation.

Keras in Python

```
import numpy as np
import matplotlib.pyplot as plt
from keras.datasets import mnist
```

```
## Using TensorFlow backend.
```

```
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.utils import np_utils
```

Keras - load data

```
nb_classes = 10
(X_train, y_train), (X_test, y_test) = mnist.load_data()
print("X_train original shape", X_train.shape)
```

```
## ('X_train original shape', (60000, 28, 28))
```

```
print("y_train original shape", y_train.shape)
```

```
## ('y_train original shape', (60000,))
```

Keras - data preprocessing

```
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
print("Training matrix shape", X_train.shape)
```

```
## ('Training matrix shape', (60000, 784))
```

```
print("Testing matrix shape", X_test.shape)
```

```
## ('Testing matrix shape', (10000, 784))
```

Keras - build model

```
model = Sequential()
model.add(Dense(512, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dropout(0.2))
# Dropout helps protect the model from memorizing
# or "overfitting" the training data
model.add(Dense(10))
model.add(Activation('softmax'))
# This special "softmax" activation among other things
# ensures the output is a valid probaility distribution
```

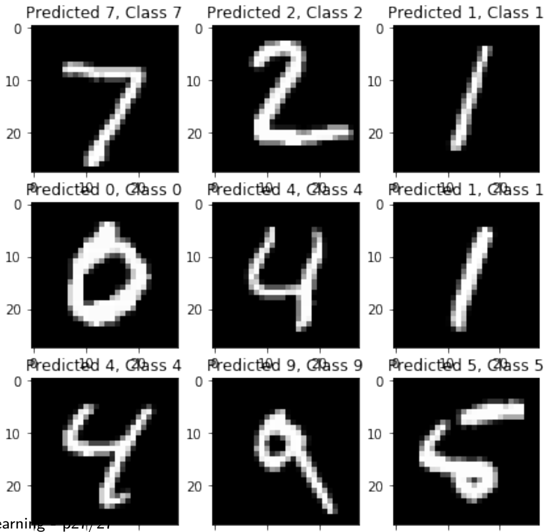
Keras - train the model

```
model.compile(loss='categorical_crossentropy',
        optimizer='adam',metrics=['accuracy'])
model.fit(X_train, Y_train,
          batch_size=128, nb_epoch=4,
          verbose=0,
          validation_data=(X_test, Y_test))
```

Keras - evaluate the model

```
score = model.evaluate(X_test, Y_test, verbose=0)
print(score[1])
```

```
## 0.9805
```



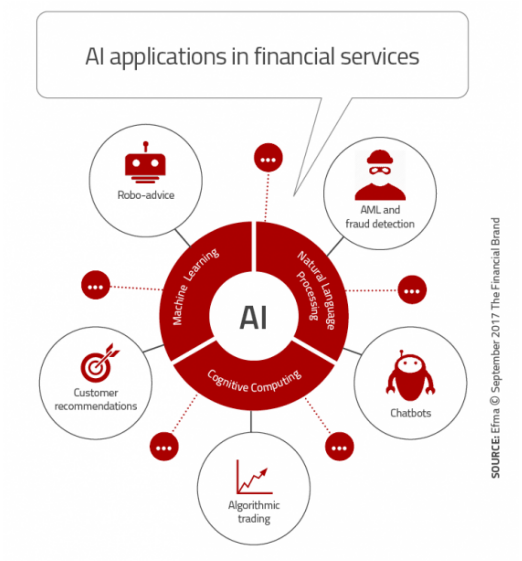
Machine Learning for Finance

Deep Neural Networks

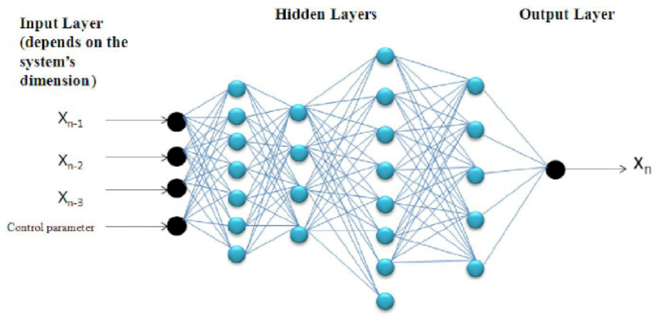
Dr Ning Wang

2019

Deep Learning in Finance



Deep Neural Networks (DNNs)



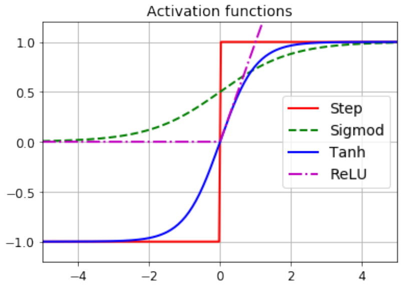
- DNN is ANN with multiple hidden layers of units between the input and output layers.
- DNNs can model complex non-linear relationships.
- DNNs are designed as feedforward and backpropagation networks.

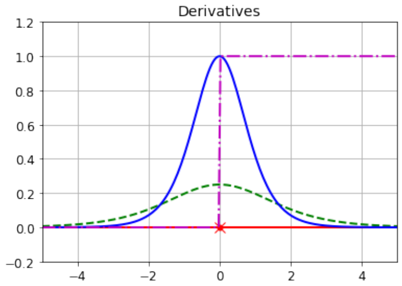
Vanishing/Exploding Gradients Problem

- X. Glorot and Y. Bengio found this problem in 2010 in paper "Understanding the difficulty of training deep feedforward neural networks".
- For DNNs gradients often get vanishing/exploding as the algorithm progresses down to the many layers.
- For RNN long term dependencies can not be efficiently captured.

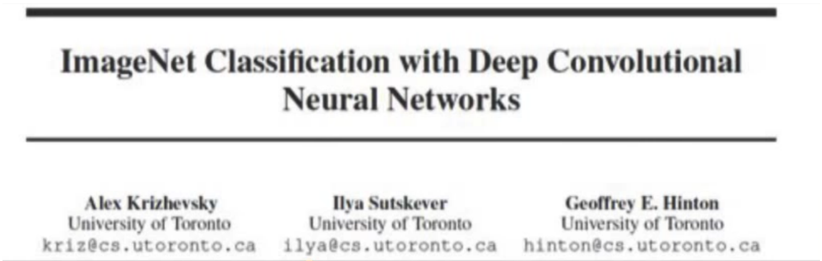
Activation functions

- The sigmoid function is continuous and differentiable but it saturates at 0 or 1, with a derivative extremely close to 0.
- The ReLU function suffers from a problem known as the dying ReLUs, i.e. some neurons stop outputting anything rather than 0.
- The variant of the ReLU function is used to sovle the gradient problems, such as $LeakyReLU_a(z) = max(az, z)$.





Convolutional Neural Network (CNN)

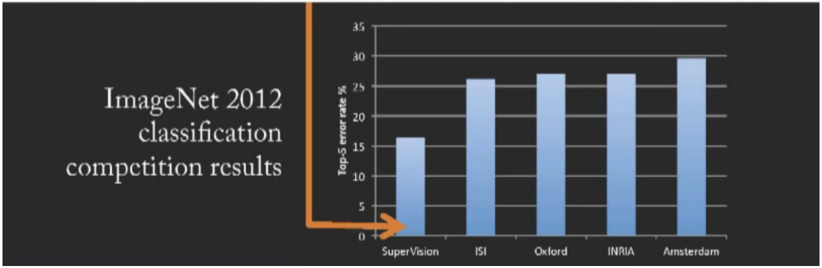


ImageNet Classification with Deep Convolutional Neural Networks

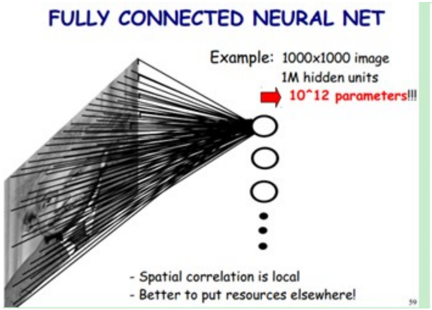
Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

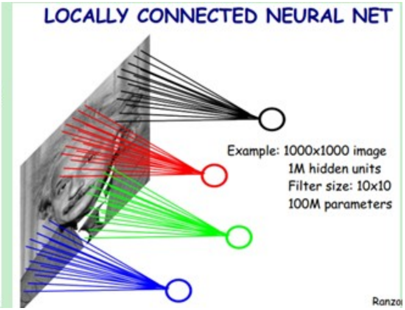
Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca



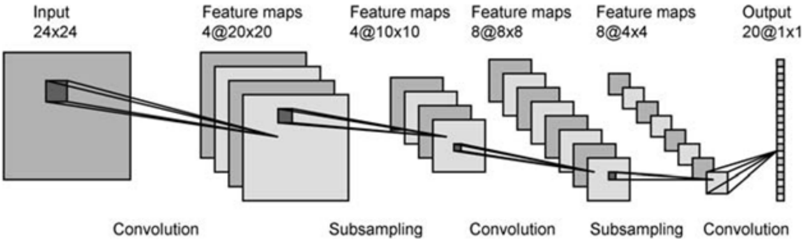
Problem for Image Recognition


FULLY CONNECTED NEURAL NET
Example: 1000x1000 image
1M hidden units
10^12 parameters!!!
- Spatial correlation is local
- Better to put resources elsewhere!
59



Convolutional Neural Network (CNN)

- CNN is a type of feed-forward artificial neural network.
- The connectivity pattern between its neurons is inspired by the organization of the animal visual cortex.
- CNN is a variation of multilayer perceptrons designed to use minimal amounts of preprocessing.
- CNN has wide applications in image and video recognition, recommender systems and natural language processing.



Convolutional Neural Network (CNN)

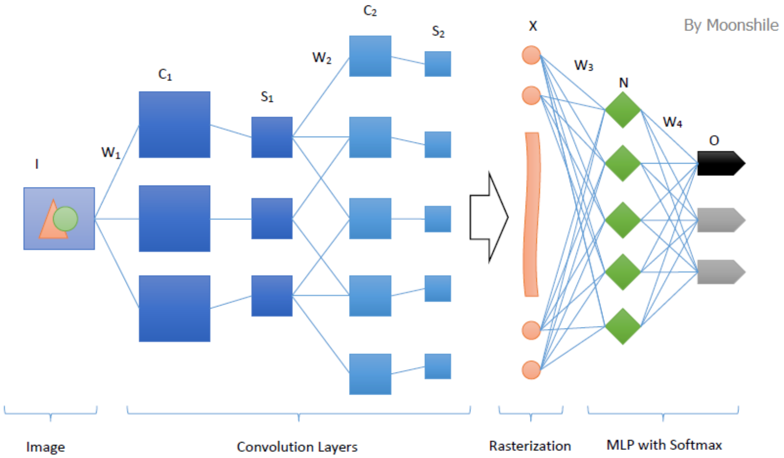
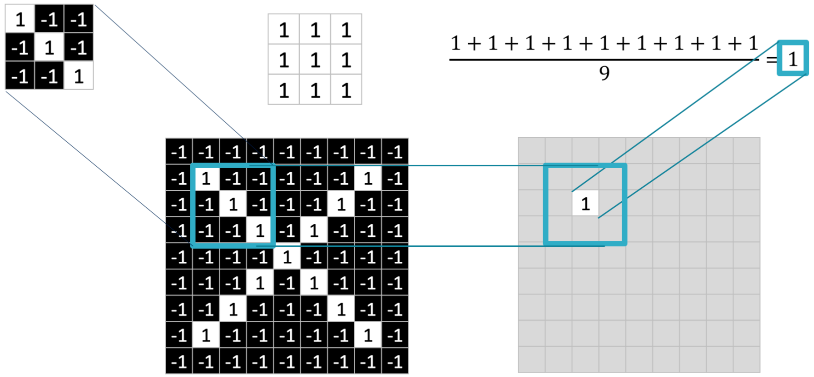
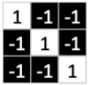


Fig. 1 CNN Structure

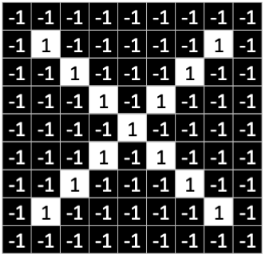
Convolutional Neural Network (CNN)



Convolutional Neural Network (CNN)

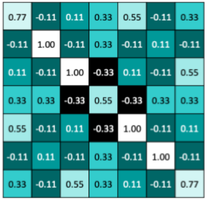


1	-1	-1
-1	1	-1
-1	-1	1



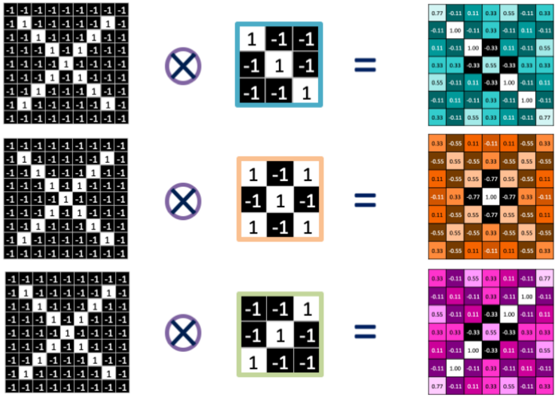
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

$\longrightarrow$



0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

Convolutional Neural Network (CNN)

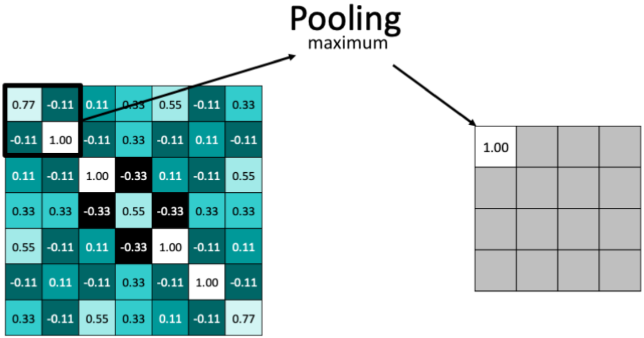


$$\begin{bmatrix} -1&-1&-1&-1&-1&-1&-1&-1&-1\\ -1&1&-1&-1&-1&-1&-1&1&-1\\ -1&-1&1&-1&-1&-1&1&-1&-1\\ -1&-1&-1&1&-1&1&-1&-1&-1\\ -1&-1&-1&-1&1&-1&-1&-1&-1\\ -1&-1&-1&1&-1&1&-1&-1&-1\\ -1&-1&1&-1&-1&-1&1&-1&-1\\ -1&1&-1&-1&-1&-1&-1&1&-1\\ -1&-1&-1&-1&-1&-1&-1&-1&-1 \end{bmatrix} \otimes \begin{bmatrix} 1&-1&-1\\ -1&1&-1\\ -1&-1&1 \end{bmatrix} = \begin{bmatrix} 0.77&-0.11&0.11&0.33&0.55&-0.11&0.33\\ -0.11&1.00&-0.11&0.33&-0.11&0.11&-0.11\\ 0.11&-0.11&1.00&-0.33&0.11&-0.11&0.55\\ 0.33&0.33&-0.33&0.55&-0.33&0.33&0.33\\ 0.55&-0.11&0.11&-0.33&1.00&-0.11&0.11\\ -0.11&0.11&-0.11&0.33&-0.11&1.00&-0.11\\ 0.33&-0.11&0.55&0.33&0.11&-0.11&0.77 \end{bmatrix}$$

$$\begin{bmatrix} -1&-1&-1&-1&-1&-1&-1&-1&-1\\ -1&1&-1&-1&-1&-1&-1&1&-1\\ -1&-1&1&-1&-1&-1&1&-1&-1\\ -1&-1&-1&1&-1&1&-1&-1&-1\\ -1&-1&-1&-1&1&-1&-1&-1&-1\\ -1&-1&-1&1&-1&1&-1&-1&-1\\ -1&-1&1&-1&-1&-1&1&-1&-1\\ -1&1&-1&-1&-1&-1&-1&1&-1\\ -1&-1&-1&-1&-1&-1&-1&-1&-1 \end{bmatrix} \otimes \begin{bmatrix} 1&-1&1\\ -1&1&-1\\ 1&-1&1 \end{bmatrix} = \begin{bmatrix} 0.33&-0.55&0.11&-0.11&0.11&-0.55&0.33\\ -0.55&0.55&-0.55&0.33&-0.55&0.55&-0.55\\ 0.11&-0.55&0.55&-0.77&0.55&-0.55&0.11\\ -0.11&0.33&-0.77&1.00&-0.77&0.33&-0.11\\ 0.11&-0.55&0.55&-0.77&0.55&-0.55&0.11\\ -0.55&0.55&-0.55&0.33&-0.55&0.55&-0.55\\ 0.33&-0.55&0.11&-0.11&0.11&-0.55&0.33 \end{bmatrix}$$

$$\begin{bmatrix} -1&-1&-1&-1&-1&-1&-1&-1&-1\\ -1&1&-1&-1&-1&-1&-1&1&-1\\ -1&-1&1&-1&-1&-1&1&-1&-1\\ -1&-1&-1&1&-1&1&-1&-1&-1\\ -1&-1&-1&-1&1&-1&-1&-1&-1\\ -1&-1&-1&1&-1&1&-1&-1&-1\\ -1&-1&1&-1&-1&-1&1&-1&-1\\ -1&1&-1&-1&-1&-1&-1&1&-1\\ -1&-1&-1&-1&-1&-1&-1&-1&-1 \end{bmatrix} \otimes \begin{bmatrix} -1&-1&1\\ -1&1&-1\\ 1&-1&-1 \end{bmatrix} = \begin{bmatrix} 0.33&-0.11&0.55&0.33&0.11&-0.11&0.77\\ -0.11&0.11&-0.11&0.33&-0.11&1.00&-0.11\\ 0.55&-0.11&0.11&-0.33&1.00&-0.11&0.11\\ 0.33&0.33&-0.33&0.55&-0.33&0.33&0.33\\ 0.11&-0.11&1.00&-0.33&0.11&-0.11&0.55\\ -0.11&1.00&-0.11&0.33&-0.11&0.11&-0.11\\ 0.77&-0.11&0.11&0.33&0.55&-0.11&0.33 \end{bmatrix}$$

Convolutional Neural Network (CNN)



0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

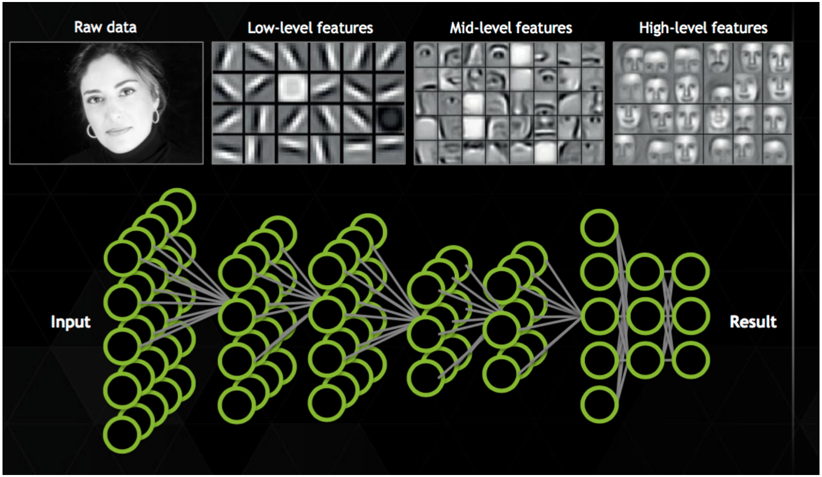
Convolutional Neural Network (CNN)

$$\begin{bmatrix} 0.77 & -0.11 & 0.11 & 0.33 & 0.55 & -0.11 & 0.33 \\ -0.11 & 1.00 & -0.11 & 0.33 & -0.11 & 0.11 & -0.11 \\ 0.11 & -0.11 & 1.00 & -0.33 & 0.11 & -0.11 & 0.55 \\ 0.33 & 0.33 & -0.33 & 0.55 & -0.33 & 0.33 & 0.33 \\ 0.55 & -0.11 & 0.11 & -0.33 & 1.00 & -0.11 & 0.11 \\ -0.11 & 0.11 & -0.11 & 0.33 & -0.11 & 1.00 & -0.11 \\ 0.33 & -0.11 & 0.55 & 0.33 & 0.11 & -0.11 & 0.77 \end{bmatrix} \longrightarrow \begin{bmatrix} 1.00 & 0.33 & 0.55 & 0.33 \\ 0.33 & 1.00 & 0.33 & 0.55 \\ 0.55 & 0.33 & 1.00 & 0.11 \\ 0.33 & 0.55 & 0.11 & 0.77 \end{bmatrix}$$

$$\begin{bmatrix} 0.33 & -0.55 & 0.11 & -0.11 & 0.11 & -0.55 & 0.33 \\ -0.55 & 0.55 & -0.55 & 0.33 & -0.55 & 0.55 & -0.55 \\ 0.11 & -0.55 & 0.55 & -0.77 & 0.55 & -0.55 & 0.11 \\ -0.11 & 0.33 & -0.77 & 1.00 & -0.77 & 0.33 & -0.11 \\ 0.11 & -0.55 & 0.55 & -0.77 & 0.55 & -0.55 & 0.11 \\ -0.55 & 0.55 & -0.55 & 0.33 & -0.55 & 0.55 & -0.55 \\ 0.33 & -0.55 & 0.11 & -0.11 & 0.11 & -0.55 & 0.33 \end{bmatrix} \longrightarrow \begin{bmatrix} 0.55 & 0.33 & 0.55 & 0.33 \\ 0.33 & 1.00 & 0.55 & 0.11 \\ 0.55 & 0.55 & 0.55 & 0.11 \\ 0.33 & 0.11 & 0.11 & 0.33 \end{bmatrix}$$

$$\begin{bmatrix} 0.33 & -0.11 & 0.55 & 0.33 & 0.11 & -0.11 & 0.77 \\ -0.11 & 0.11 & -0.11 & 0.33 & -0.11 & 1.00 & -0.11 \\ 0.55 & -0.11 & 0.11 & -0.33 & 1.00 & -0.11 & 0.11 \\ 0.33 & 0.33 & -0.33 & 0.55 & -0.33 & 0.33 & 0.33 \\ 0.11 & -0.11 & 1.00 & -0.33 & 0.11 & -0.11 & 0.55 \\ -0.11 & 1.00 & -0.11 & 0.33 & -0.11 & 0.11 & -0.11 \\ 0.77 & -0.11 & 0.11 & 0.33 & 0.55 & -0.11 & 0.33 \end{bmatrix} \longrightarrow \begin{bmatrix} 0.33 & 0.55 & 1.00 & 0.77 \\ 0.55 & 0.55 & 1.00 & 0.33 \\ 1.00 & 1.00 & 0.11 & 0.55 \\ 0.77 & 0.33 & 0.55 & 0.33 \end{bmatrix}$$

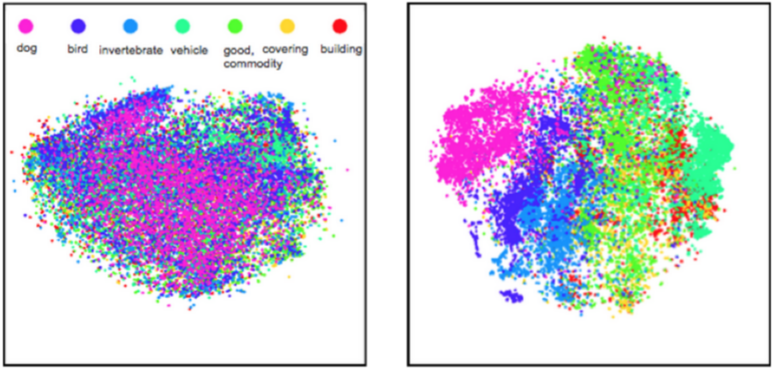
Convolutional Neural Network (CNN)



Convolutional Neural Network (CNN)

Why Deep Learning?

The Unreasonable Effectiveness of Deep Features



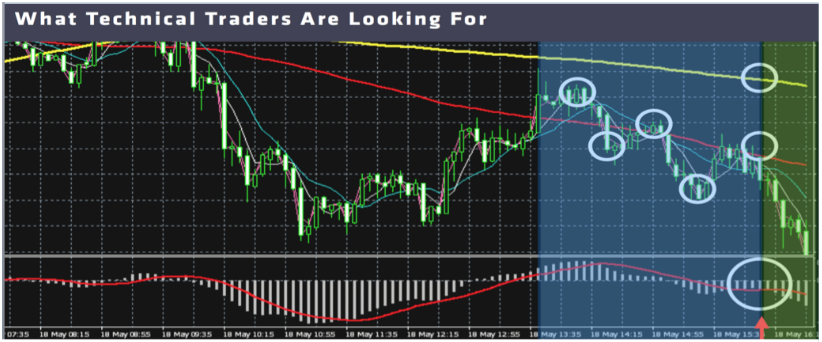
Low-level: Pool_1 High-level: FC_6

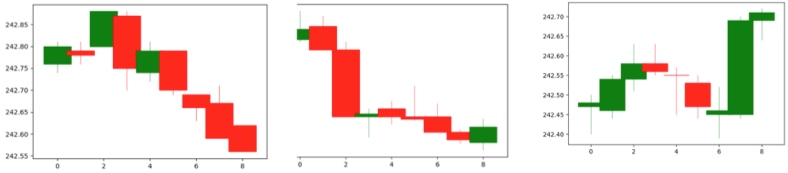
Classes separate in the deep representations and transfer to many tasks.
[DeCAF] [Zeiler-Fergus]

CNN in Keras

```
from keras.layers import Convolution1D, MaxPooling1D
model = Sequential()
model.add(Convolution1D(input_shape = TRAIN_SIZE,
                        nb_filter=64,
                        filter_length=2,
                        border_mode='valid',
                        activation='relu',
                        subsample_length=1))
model.add(MaxPooling1D(pool_length=2))
model.add(Dense(250))
model.add(Dropout(0.25))
model.add(Activation('relu'))
```

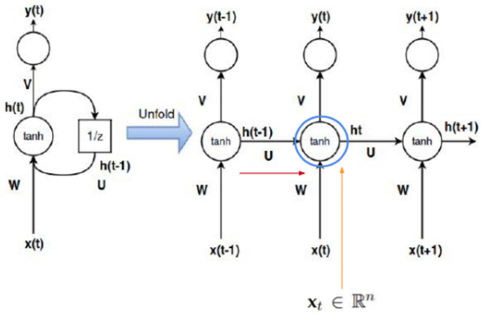
CNN Example


What Technical Traders Are Looking For
07:35
18 May 08:15
18 May 08:55
18 May 09:35
18 May 10:15
18 May 10:55
18 May 11:35
18 May 12:15
18 May 12:55
18 May 13:35
18 May 14:15
18 May 14:55
18 May 15:35
18 May 16:15


242.85
242.80
242.75
242.70
242.65
242.60
242.55
0
2
4
6
8
242.70
242.65
242.60
242.55
242.50
242.45
242.40

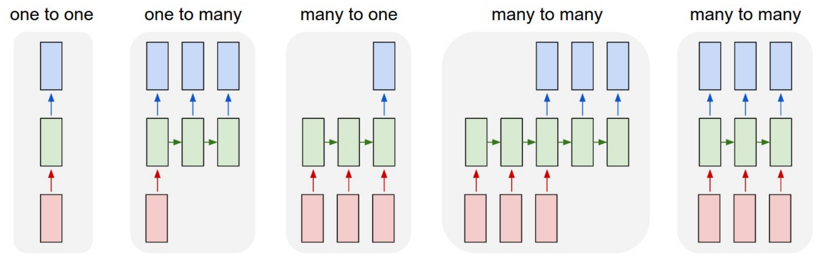
Recurrent Neural Network (RNN)

- RNN is a class of artificial neural network where connections between units form a directed cycle.
- Unlike feed-forward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs.
- RNN is applicable to tasks such as sequential data processing.



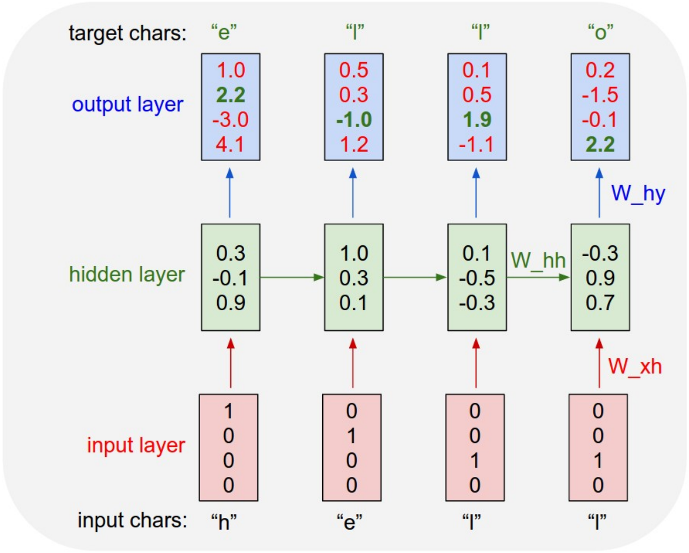
$$\mathbf{h}_t = g(\mathbf{W} \cdot \mathbf{x}_t + \mathbf{U} \cdot \mathbf{h}_{t-1} + \mathbf{b}_h)$$

Recurrent Neural Network (RNN)[1]



[1] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

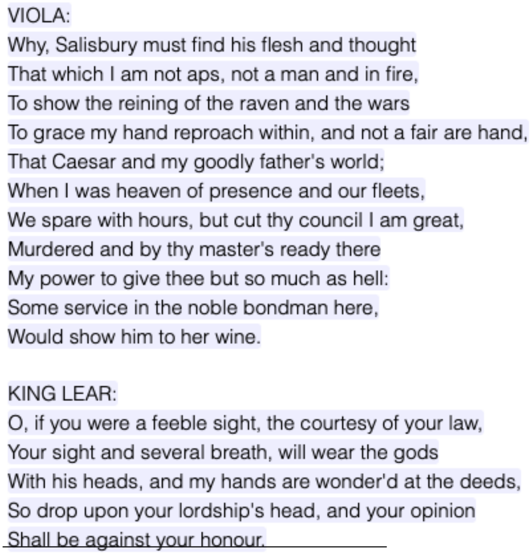
Recurrent Neural Network (RNN)[2]



[2] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

RNN Example - Shakespeare [3]

Train with 3-layer RNN with 512 hidden nodes on each layer

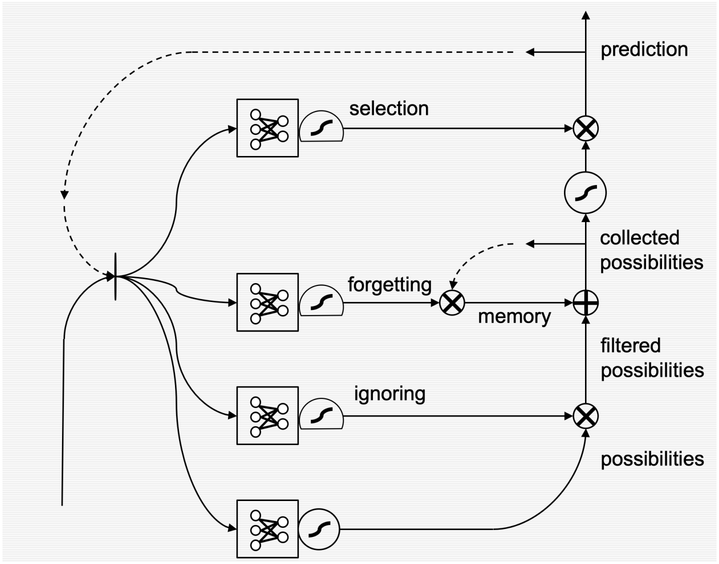


VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.

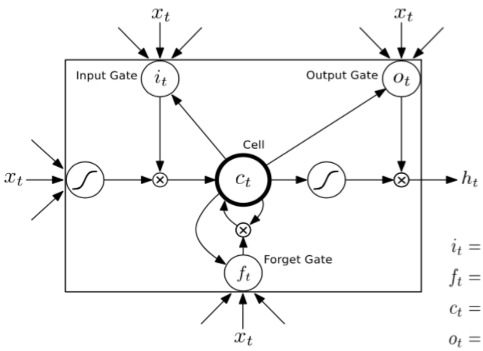
[3] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Long-Short-Term-Memory in RNN[4]



[4] Cristopher Olah, "Understanding LSTM Networks" (2015)

Long-Short-Term-Memory in RNN[5]



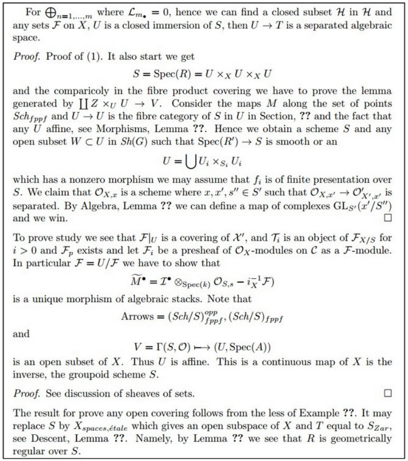
$$
\begin{aligned}
i_t &= \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right)\\
f_t &= \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)\\
c_t &= f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right)\\
o_t &= \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right)\\
h_t &= o_t \tanh(c_t)
\end{aligned}
$$

[5]Cristopher Olah, "Understanding LSTM Networks" (2015)

LSTM in Keras

```
from keras.layers.recurrent import LSTM
model = Sequential()
model.add(LSTM(input_shape = (EMB_SIZE,),
          input_dim=EMB_SIZE, output_dim=HIDDEN_RNN,
          return_sequences=True))
model.add(Dense(1))
model.add(Activation('linear'))
```

LSTM Example - Algebraic Geometry [6]



For $\bigoplus_{n=1,\dots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \operatorname{Spec}(R) = U \times_X U \times_X U$$

and the comparicoly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, **??** and the fact that any U affine, see Morphisms, Lemma **??**. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\operatorname{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma **??** we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\operatorname{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

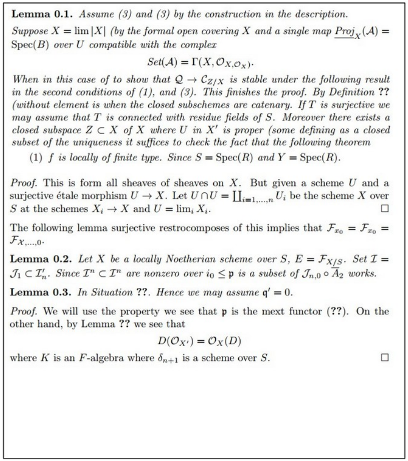
and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \operatorname{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example **??**. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma **??**. Namely, by Lemma **??** we see that R is geometrically regular over S.



Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ (by the formal open covering X and a single map $\underline{Proj}_X(\mathcal{A}) = \operatorname{Spec}(B)$ over U compatible with the complex

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

*When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition **??** (without element is when the closed subschemes are catenary. If T is surjective we may assume that T is connected with residue fields of S. Moreover there exists a closed subspace $Z \subset X$ of X where U in X' is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem*

(1) *f is locally of finite type. Since $S = \operatorname{Spec}(R)$ and $Y = \operatorname{Spec}(R)$.*

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\dots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\dots,0}$.

Lemma 0.2. *Let X be a locally Noetherian scheme over S, $E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. Since $\mathcal{I}^n \subset \mathcal{I}^n$ are nonzero over $i_0 \leq \mathfrak{p}$ is a subset of $\mathcal{J}_{n,0} \circ \overline{A}_2$ works.*

Lemma 0.3. *In Situation **??**. Hence we may assume $\mathfrak{q}' = 0$.*

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (**??**). On the other hand, by Lemma **??** we see that

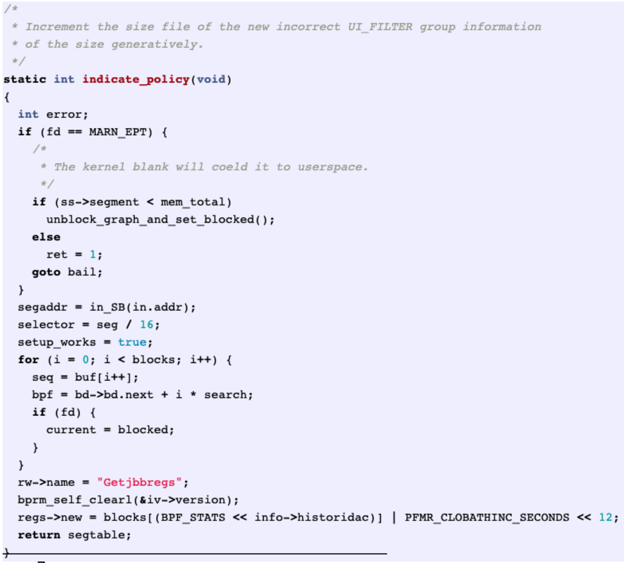
$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □

[6] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

LSTM Example - Linux Source Code [7]

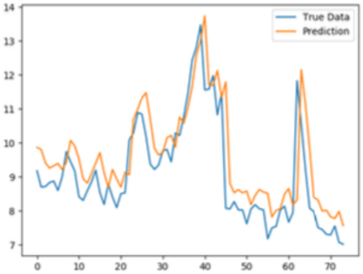
Train 474MB of C code with 3-layer LSTMs



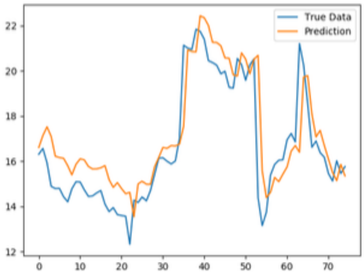
```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

[7] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

LSTM Example - Time series



(a) Test Data SPY: Feature-Selected LSTM



(b) Test Data AAPL: Feature-Selected Vanilla RNN

Figure 3.5: Predictions vs True Data for feature-selected models

Further Readings

- *Deep Learning* By Ian Goodfellow and Yoshua Bengio and Aaron Courville MIT Press, 2016
- *Neural Networks and Deep Learning* By Michael Nielsen, Online book, 2016
- *Learning Deep Architectures for AI* By Yoshua Bengio, NOW Publishers, 2009