MSc Data Mining

Topic 06 : Data Modelling - Advanced

Part 01 : Data Modelling - Advanced

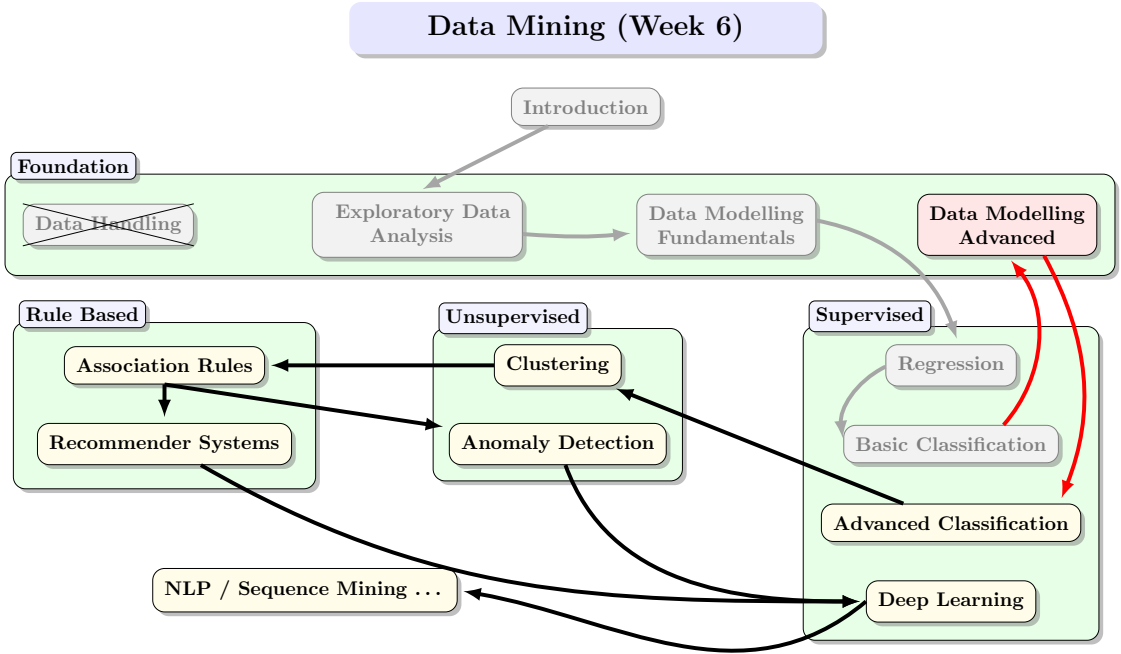
Dr Bernard Butler and Dr Kieran Murphy

Department of Computing and Mathematics, WIT.
(bernard.butler@wit.ie; kmurphy@wit.ie)

Spring Semester, 2022

Outline

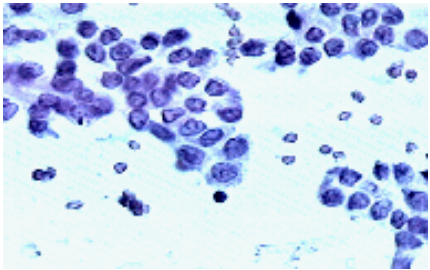
- Feature engineering/selection
- Hyperparemeter tuning
- Ensemble Learning
- Explainable AI (XAI)


Data Mining (Week 6)
Introduction
Foundation
Data Handling
Exploratory Data Analysis
Data Modelling Fundamentals
Data Modelling Advanced
Rule Based
Association Rules
Recommender Systems
Unsupervised
Clustering
Anomaly Detection
Supervised
Regression
Basic Classification
Advanced Classification
NLP / Sequence Mining ...
Deep Learning

Outline

Wisconsin Dataset — Breast Cancer (WDBC)*

Outline



- Aims to predict breast cancer diagnosis based on Fine Needle Aspiration (FNA).
- Resulting classifier using on these nine features successfully diagnosed 97% of new cases.

Construction

- FNA's were done on a total of 569 patients, samples were stained to help differentiate distinguished cell nuclei
- Samples were classified as cancer-based through biopsy and historical confirmation. Non-cancer samples were confirmed by biopsy or follow ups.
- Users then chose areas of the FNA with minimal overlap between nuclei; they then took scans utilising a digital camera.
- `Xcyt` was used to create approximate boundaries, which would then used a process called snakes which converged to give the exact nuclei boundary.
- Once the boundaries for the nuclei were set, calculations were made (of mean, standard error, and max) resulting in 3×10 features.

WDBC — Load Data

```
UCI = "https://archive.ics.uci.edu/ml/machine-learning-databases/"
DATA_URL = f"{UCI}/breast-cancer-wisconsin/wdbc.data"
DATA_LOCAL = "data/wdbc.data"
SEED = 42

names = ['id_number', 'diagnosis', 'radius_mean',
         'texture_mean', 'perimeter_mean', 'area_mean',
         'smoothness_mean', 'compactness_mean', 'concavity_mean',
         'concave_points_mean', 'symmetry_mean',
         'fractal_dimension_mean', 'radius_se', 'texture_se',
         'perimeter_se', 'area_se', 'smoothness_se',
         'compactness_se', 'concavity_se', 'concave_points_se',
         'symmetry_se', 'fractal_dimension_se',
         'radius_worst', 'texture_worst', 'perimeter_worst',
         'area_worst', 'smoothness_worst',
         'compactness_worst', 'concavity_worst',
         'concave_points_worst', 'symmetry_worst',
         'fractal_dimension_worst']

df = pd.read_csv(DATA_URL,header=None, names=names)
df.head(10)
```

WDBC — Load Data

```
UCI = "https://archive.ics.uci.edu/ml/machine-learning-databases/"
DATA_URL = f"{UCI}/breast-cancer-wisconsin/wdbc.data"
DATA_LOCAL = "data/wdbc.data"
SEED = 42
```

	id_number	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave_points_mean	.
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	.
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	.
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	.
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	.
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	.
5	843786	M	12.45	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.08089	.
6	844359	M	18.25	19.98	119.60	1040.0	0.09463	0.10900	0.11270	0.07400	.
7	84458202	M	13.71	20.83	90.20	577.9	0.11890	0.16450	0.09366	0.05985	.
8	844981	M	13.00	21.82	87.50	519.8	0.12730	0.19320	0.18590	0.09353	.
9	84501001	M	12.46	24.04	83.97	475.9	0.11860	0.23960	0.22730	0.08543	.

10 rows × 32 columns

```
df = pd.read_csv(DATA_URL,header=None, names=names)
df.head(10)
```

WDBC — View Data

See EDA of Breast Cancer Dataset for EDA of this dataset.

	count	mean	std	min	25%	50%	75%	max
id_number	569.0	3.037183e+07	1.250206e+08	8670.000000	869218.000000	906024.000000	8.813129e+06	9.113205e+08
radius_mean	569.0	1.412729e+01	3.524049e+00	6.981000	11.700000	13.370000	1.578000e+01	2.811000e+01
texture_mean	569.0	1.928965e+01	4.301036e+00	9.710000	16.170000	18.840000	2.180000e+01	3.928000e+01
perimeter_mean	569.0	9.196903e+01	2.429898e+01	43.790000	75.170000	86.240000	1.041000e+02	1.885000e+02
area_mean	569.0	6.548891e+02	3.519141e+02	143.500000	420.300000	551.100000	7.827000e+02	2.501000e+03
smoothness_mean	569.0	9.636028e-02	1.406413e-02	0.052630	0.086370	0.095870	1.053000e-01	1.634000e-01
compactness_mean	569.0	1.043410e-01	5.281276e-02	0.019380	0.064920	0.092630	1.304000e-01	3.454000e-01
concavity_mean	569.0	8.879932e-02	7.971981e-02	0.000000	0.029560	0.061540	1.307000e-01	4.268000e-01
concave_points_mean	569.0	4.891915e-02	3.880284e-02	0.000000	0.020310	0.033500	7.400000e-02	2.012000e-01
symmetry_mean	569.0	1.811619e-01	2.741428e-02	0.106000	0.161900	0.179200	1.957000e-01	3.040000e-01
fractal_dimension_mean	569.0	6.279761e-02	7.060363e-03	0.049960	0.057700	0.061540	6.612000e-02	9.744000e-02
radius_se	569.0	4.051721e-01	2.773127e-01	0.111500	0.232400	0.324200	4.789000e-01	2.873000e+00
texture_se	569.0	1.216853e+00	5.516484e-01	0.360200	0.833900	1.108000	1.474000e+00	4.885000e+00
perimeter_se	569.0	2.866059e+00	2.021855e+00	0.757000	1.606000	2.287000	3.357000e+00	2.198000e+01
area_se	569.0	4.033708e+01	4.549101e+01	6.802000	17.850000	24.530000	4.519000e+01	5.422000e+02
smoothness_se	569.0	7.040979e-03	3.002518e-03	0.001713	0.005169	0.006380	8.146000e-03	3.113000e-02
compactness_se	569.0	2.547814e-02	1.790818e-02	0.002252	0.013080	0.020450	3.245000e-02	1.354000e-01
concavity_se	569.0	3.189372e-02	3.018606e-02	0.000000	0.015090	0.025890	4.205000e-02	3.960000e-01
concave_points_se	569.0	1.179614e-02	6.170285e-03	0.000000	0.007638	0.010930	1.471000e-02	5.279000e-02
symmetry_se	569.0	2.054230e-02	8.266372e-03	0.007882	0.015160	0.018730	2.348000e-02	7.895000e-02
fractal_dimension_se	569.0	3.794904e-03	2.646071e-03	0.000895	0.002248	0.003187	4.558000e-03	2.984000e-02
radius_worst	569.0	1.626919e+01	4.833242e+00	7.930000	13.010000	14.970000	1.879000e+01	3.604000e+01
texture_worst	569.0	2.567722e+01	6.146258e+00	12.020000	21.080000	25.410000	2.972000e+01	4.954000e+01
perimeter_worst	569.0	1.072612e+02	3.360254e+01	50.410000	84.110000	97.660000	1.254000e+02	2.512000e+02
area_worst	569.0	8.805831e+02	5.693570e+02	185.200000	515.300000	686.500000	1.084000e+03	4.254000e+03

- no missing values
- big differences in mean/std
  $\Rightarrow$ need normalising.
- 30 numerical features, some of which expect to be correlated
  $\Rightarrow$ use PCA.

WDBC — Prepare Data

- Extract feature matrix and target column — the first two columns are case number `id_number` and target `diagnosis`.

```
3 X = df.iloc[:, 2:].values
  y = df.diagnosis.values
```

- Encode (categorical) target column

```
4 print (y[:20])

  from sklearn.preprocessing import LabelEncoder
  le = LabelEncoder()
  y = le.fit_transform(y)

  print (le.transform(["M","B"]))
  print (y[:20])
```

```
['M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M' 'M'
 'M' 'B']
[1 0]
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
```

Now have 2D array of features, `X`, and 1D array (vector) of targets, `y`.

Titanic dataset

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C
10	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	G6	S
11	12	1									C103	S
12	13	0									NaN	S
13	14	0	3	Andersson, Mr. Anders Johan	male	39.0	1	5	347082	31.2750	NaN	S
14	15	0	3	Vestrom, Miss. Hulda Amanda Adolfina	female	14.0	0	0	350406	7.8542	NaN	S
15	16	1	2	Hewlett, Mrs. (Mary D Kingcome)	female	55.0	0	0	248706	16.0000	NaN	S
16	17	0	3	Rice, Master. Eugene	male	2.0	4	1	382652	29.1250	NaN	Q
17	18	1	2	Williams, Mr. Charles Eugene	male	NaN	0	0	244373	13.0000	NaN	S
18	19	0	3	Vander Planke, Mrs. Julius (Emelia Maria Vande...	female	31.0	1	0	345763	18.0000	NaN	S
19	20	1	3	Masselmani, Mrs. Fatima	female	NaN	0	0	2649	7.2250	NaN	C
20	21	0	2	Fynney, Mr. Joseph J	male	35.0	0	0	239865	26.0000	NaN	S

How well can we predict a passenger's survival using information at time of departure?

Titanic — load

- Dataset is split into two parts:
  - `train.csv` — 891 rows with `Survived` column, used in EDA and model training.
  - `test.csv` — 418 rows without the `Survived` column, used in competition scoring.

5
```
df = pd.read_csv("train.csv")
print(df.shape)
df.head(25)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embark
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

(See Week 2)

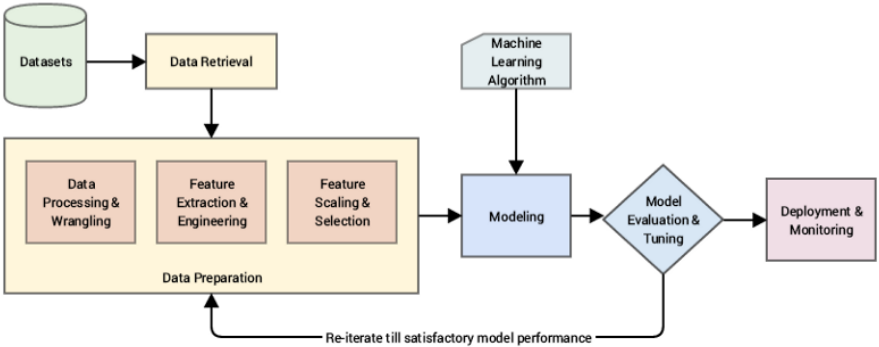
- We could convert `Sex` or `Embarked`, to a category, but since their levels are not ordered there is no big advantage.
- We don't want to convert `Name`, `Ticket` and `Cabin` since we want to perform further text processing on these columns. For example, extracting title (Capt, Mr, Miss, etc.) out of `Name`.
- We have missing values (**that are plausibly linked to target**) that we need to deal with.

6
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
```

Outline

Data Pipeline (again)



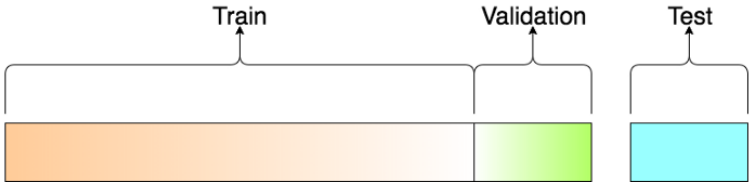
We want to make decisions on:

- Feature engineering
- Model selection
- Hyper-parameter tuning

How can we deal with multiple decisions?

- Our train/test split is useful but it does not go far enough.
- Switch to train / validation / test split
- Or better, use K-Fold Cross Validation

Train / Validation / Test Split



train dataset is used to train the model. The model sees and learns from this data.

- Since the model has learnt from this dataset it is unsuitable for generating unbiased evaluation estimates.

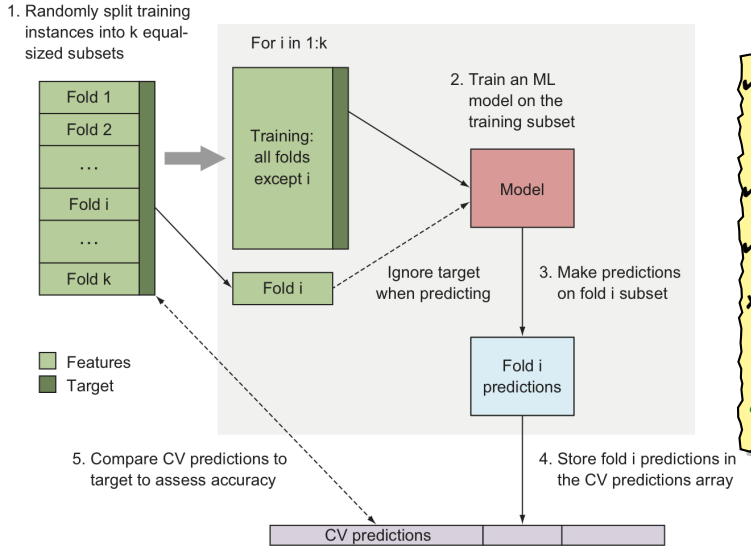
validation dataset is used tune the hyper-parameters in the model.

- The model has not seen this data during training but does see it while tuning hyper-parameters so depending on the degree of tuning, evaluation estimates will become more biased.

test dataset is used to evaluate the model.

- The model has not seen this data, so it can provide unbiased evaluation estimates.

K-Fold Cross Validation



- ✔ No need for train / validation split as fold i is the validation dataset while other $k-1$ folds are the training dataset.
- ✔ Can repeatedly apply CV, for separate decisions.
- ✔ sklearn has comprehensive support for cross validation.
- ✘ Computational effort increased by factor of k, but is easily parallelised, use option `n_jobs=-1`.
- • Typical value for k are 5, 10, or 20.

WDBC — Approach 1 — Training using Train/Test Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, random_state=SEED)

from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X_train_scaled = ss.fit_transform(X_train)
X_test_scaled = ss.transform(X_test)

from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='lbfgs')
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_pred)
print('accuracy score: %s' % score)
```

WDBC — Approach 1 — Training using Train/Test Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, random_state=SEED)

from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X_train_scaled = ss.fit_transform(X_train)
X_test_scaled = ss.transform(X_test)

from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='lbfgs')
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_pred)
print('accuracy score: %s' % score)
```

- Split into train/test subsets . . .

WDBC — Approach 1 — Training using Train/Test Split

```
9
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, random_state=SEED)

from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X_train_scaled = ss.fit_transform(X_train)
X_test_scaled = ss.transform(X_test)

from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='lbfgs')
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_pred)
print('accuracy score: %s' % score)
```

- Split into train/test subsets . . . normalise . . .

WDBC — Approach 1 — Training using Train/Test Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, random_state=SEED)

from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X_train_scaled = ss.fit_transform(X_train)
X_test_scaled = ss.transform(X_test)

from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='lbfgs')
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_pred)
print('accuracy score: %s' % score)
```

- Split into train/test subsets . . . normalise . . . and train/fit . . .

WDBC — Approach 1 — Training using Train/Test Split

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, random_state=SEED)

from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X_train_scaled = ss.fit_transform(X_train)
X_test_scaled = ss.transform(X_test)

from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='lbfgs')
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_pred)
print('accuracy score: %s' % score)
```

- Split into train/test subsets . . . normalise . . . and train/fit . . . predict . . .

WDBC — Approach 1 — Training using Train/Test Split

```
12
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, random_state=SEED)

from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X_train_scaled = ss.fit_transform(X_train)
X_test_scaled = ss.transform(X_test)

from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='lbfgs')
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_pred)
print('accuracy score: %s' % score)
```

```
accuracy score: 0.9824561403508771
```

- Split into train/test subsets … normalise … and train/fit … predict … evaluate.

WDBC — Approach 1 — Training using Train/Test Split

```
13 from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, random_state=SEED)

from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X_train_scaled = ss.fit_transform(X_train)
X_test_scaled = ss.transform(X_test)

from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='lbfgs')
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)

from sklearn.metrics import accuracy_score
score = accuracy_score(y_test, y_pred)
print('accuracy score: %s' % score)
```

```
accuracy score: 0.9824561403508771
```

- Split into train/test subsets … normalise … and train/fit … predict … evaluate.

If I make modelling decisions based on the test score, then I can not longer treat its estimates as unbiased.

WDBC — Approach 2 — Training using Cross Validation (CV)

```
14 from sklearn.model_selection import train_test_split
   X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, random_state=SEED)

   from sklearn.preprocessing import StandardScaler
   ss = StandardScaler()
   X_train_scaled = ss.fit_transform(X_train)
   X_test_scaled = ss.transform(X_test)

   from sklearn.linear_model import LogisticRegression
   model = LogisticRegression(solver='lbfgs')

   from sklearn.model_selection import cross_val_score
   scores = cross_val_score(model, X=X_train_scaled, y=y_train, cv=10, n_jobs=-1)

   print('CV accuracy scores: %s' % scores)
   print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

- Function `cross_val_score` takes in the model, the data (features and targets), and cross validation parameters. Then applies CV process, returning sequence of scores for each iteration of the CV.
- Cross validation is easily parallelised — use option `n_jobs=-1` for all cores.

WDBC — Approach 2 — Training using Cross Validation (CV)

```
15 from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, random_state=SEED)

from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X_train_scaled = ss.fit_transform(X_train)
X_test_scaled = ss.transform(X_test)

from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='lbfgs')

from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X=X_train_scaled, y=y_train, cv=10, n_jobs=-1)

print('CV accuracy scores: %s' % scores)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

- Split into train/test subsets . . .
- Function `cross_val_score` takes in the model, the data (features and targets), and cross validation parameters. Then applies CV process, returning sequence of scores for each iteration of the CV.
- Cross validation is easily parallelised — use option `n_jobs=-1` for all cores.

WDBC — Approach 2 — Training using Cross Validation (CV)

```
16
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, random_state=SEED)

from sklearn.preprocessing import StandardScaler
ss = StandardScaler()
X_train_scaled = ss.fit_transform(X_train)
X_test_scaled = ss.transform(X_test)

from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='lbfgs')

from sklearn.model_selection import cross_val_score
scores = cross_val_score(model, X=X_train_scaled, y=y_train, cv=10, n_jobs=-1)

print('CV accuracy scores: %s' % scores)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

- Split into train/test subsets … normalise …
- Function `cross_val_score` takes in the model, the data (features and targets), and cross validation parameters. Then applies CV process, returning sequence of scores for each iteration of the CV.
- Cross validation is easily parallelised — use option `n_jobs=-1` for all cores.

WDBC — Approach 2 — Training using Cross Validation (CV)

```
17 from sklearn.model_selection import train_test_split
   X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, random_state=SEED)

   from sklearn.preprocessing import StandardScaler
   ss = StandardScaler()
   X_train_scaled = ss.fit_transform(X_train)
   X_test_scaled = ss.transform(X_test)

   from sklearn.linear_model import LogisticRegression
   model = LogisticRegression(solver='lbfgs')

   from sklearn.model_selection import cross_val_score
   scores = cross_val_score(model, X=X_train_scaled, y=y_train, cv=10, n_jobs=-1)

   print('CV accuracy scores: %s' % scores)
   print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

- Split into train/test subsets … normalise … and train.
- Function `cross_val_score` takes in the model, the data (features and targets), and cross validation parameters. Then applies CV process, returning sequence of scores for each iteration of the CV.
- Cross validation is easily parallelised — use option `n_jobs=-1` for all cores.

WDBC — Approach 2 — Training using Cross Validation (CV)

```
18 from sklearn.model_selection import train_test_split
   X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, random_state=SEED)

   from sklearn.preprocessing import StandardScaler
   ss = StandardScaler()
   X_train_scaled = ss.fit_transform(X_train)
   X_test_scaled = ss.transform(X_test)

   from sklearn.linear_model import LogisticRegression
   model = LogisticRegression(solver='lbfgs')

   from sklearn.model_selection import cross_val_score
   scores = cross_val_score(model, X=X_train_scaled, y=y_train, cv=10, n_jobs=-1)

   print('CV accuracy scores: %s' % scores)
   print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

```
CV accuracy scores: [0.97142857 0.97058824 0.94117647 1.   1.         1.
 0.97058824 0.91176471 0.97058824 0.97058824]
CV accuracy: 0.971 +/- 0.026
```

- Split into train/test subsets … normalise … and train.
- Function `cross_val_score` takes in the model, the data (features and targets), and cross validation parameters. Then applies CV process, returning sequence of scores for each iteration of the CV.
- Cross validation is easily parallelised — use option `n_jobs=-1` for all cores.

WDBC — Approach 3 — Using a Pipeline I

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, random_state=SEED)

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

pipeline = Pipeline([
    ('scl', StandardScaler()),
    ('clf', LogisticRegression(solver='lbfgs'))
])

scores = cross_val_score(pipeline, X=X_train, y=y_train, cv=10, n_jobs=-1)

print('CV accuracy scores: %s' % scores)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

- A pipeline is a sequence (list) of models (scaler/filters/classifiers/...) which is passed to `cross_val_score` instead of classifier as in previous slide.
- Pipes can ensure that operations (transformations, new features added) on train dataset are also applied to test/validation dataset.

WDBC — Approach 3 — Using a Pipeline I

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, random_state=SEED)

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

pipeline = Pipeline([
    ('scl', StandardScaler()),
    ('clf', LogisticRegression(solver='lbfgs'))
])

scores = cross_val_score(pipeline, X=X_train, y=y_train, cv=10, n_jobs=-1)

print('CV accuracy scores: %s' % scores)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

- A pipeline is a sequence (list) of models (scaler/filters/classifiers/...) which is passed to `cross_val_score` instead of classifier as in previous slide.
- Pipes can ensure that operations (transformations, new features added) on train dataset are also applied to test/validation dataset.

WDBC — Approach 3 — Using a Pipeline

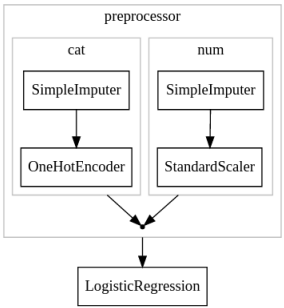

```
21 from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.6, random_state=SEED)

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

pipeline = Pipeline([
    ('scl', StandardScaler()),
    ('clf', LogisticRegression(solver='lbfgs'))
])

scores = cross_val_score(pipeline, X=X_train, y=y_train, cv=10, n_jobs=-1)

print('CV accuracy scores: %s' % scores)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

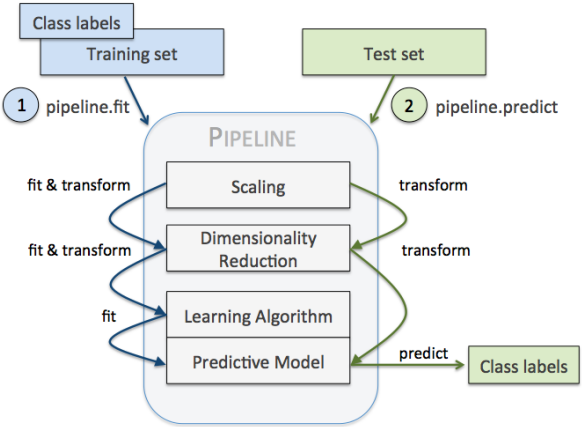
```
CV accuracy scores: [0.97826087 0.97826087 0.97826087 0.95652174 1.
1.
 0.97777778 0.97777778 0.95555556 0.93333333]
CV accuracy: 0.974 +/- 0.019
```

- A pipeline is a sequence (list) of models (scaler/filters/classifiers/...) which is passed to `cross_val_score` instead of classifier as in previous slide.
- Pipes can ensure that operations (transformations, new features added) on train dataset are also applied to test/validation dataset.

Why use Pipelines?

- Simplifies hyper-parameter tuning (next section) of intermediate models in the pipeline.
- Ensures consistent treatment of data (`fit` on train and `transform` on both train and test datasets).
- Pipeline can be split so different features are processed using different sequence of models.





Outline

What is Feature Engineering?

Feature engineering

The process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data.

- is a representation problem*

  '*you have to turn your inputs into things the algorithm can understand*'

  – Shayne Miel

- is an Art

  '*Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering.*'

  – Andrew Ng

  '*. . . some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used.*'

  – Pedro Domingos, (A Few Useful Things to Know about Machine Learning)"

Feature Engineering $\approx$ Extraction $\cup$ Engineering $\cup$ Transformations $\cup$ Selection

Why Feature Engineering?

- **Better representation of data**
  - Improved representation can be better understood by ML algorithms.
  - Better representations leads to improved visualisation — for example, compare the frequent word occurrences of a newspaper article as opposed to the raw text.
- **Better performing models**
  - The right features tend to give models that outperform other models no matter how complex the algorithm is.
  - In general if you have the right feature set, even a simple model will perform well and give desired results.

  Better features make better models.
- **Essential for model building and evaluation**
  - Numerical data is slower on decision trees.
  - Categorical data not suitable for linear regression $\Longrightarrow$ apply one-hot encoding.
- **More flexibility on data types**
  - Want to build models on diverse data types (text, images, video).
- **Emphasis on the business and domain**
  - Better features are often motivated by domain experts (not data scientists), leading to more understandable (by humans) models.
  - Feature engineering emphasises to focus on the business and the domain of the problem when building features.

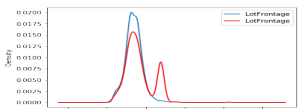
Feature Engineering Techniques

- **Imputation** — replacing missing values
- **Outliers** — identification and resolution of extreme values.
- **Binning** — condensing/compress a feature to contain less informaion
- **Log Transform** — convert skewed to closer approximate to normal
- **Categorical Encoding** — converting label to numerical values for learners
- **Normalisation and Scaling** — shift and scale features so that they can be directly compared.
- **Grouping Operations** — build features based on aggregate functions
- **Splitting** — extracting information from part of a feature
  - Example: extracting title from person's name in Titanic dataset.

Imputation

Imputation is a more preferable option rather than dropping because it preserves the data size, and can preserve location of missing values (important when location of missing values is not random or is related to target).

- Numerical imputation relies on replacing NA by typical values (mean/median), by random value, or values based on correlated columns (features) or row (observations).
- End of tail imputation replaces NA by arbitrary values from end of the variable distribution (i.e. **atypical** values) — the centre of the variable distribution is not been distorted by the missing values.



- Categorical imputation relies on replacing NA by mode, by random value, or creating a new category level.

Gender
Male
Male
NA
Female
Male
NA
Female

→

Gender
Male
Male
Missing
Female
Male
Missing
Female

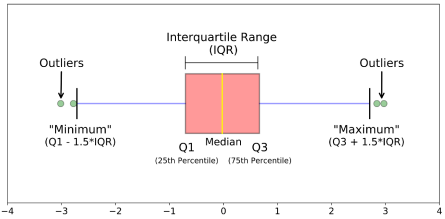
Missing Indicator Feature

A missing indicator is an additional binary variable that indicates whether the data was missing for an observation (1) or not (0). The aim is to capture where data is missing.

Outliers

Identification

- . . . via standard deviation
  - If a value has a distance to the average higher than $x\times$ standard deviation, it can be assumed as an outlier
- . . . via Percentiles (relative position / IQR)
  - Bottom $x\%$ and top $x\%$ of the observations.
  - Values based on $1.5 \times$ IQR rule.



Resolution

- **Trimming** — removing the outliers from our dataset.
- **Imputing** — treat outliers as missing data, and apply missing data imputation techniques.
- **Discretisation** – place outliers in edge bins with higher or lower values of the distribution.
- **Censoring** — cap the variable distribution at the maximum and minimum values.

Binning

- Numerical binning converts a numerical feature into an (ordered) categorical feature:
  - **Binarisation** — Generate a binary feature from a numeric feature.
  - **Rounding** — reduce the precision in the data to simplify decision tree models, or as a step towards representing as categorical.
  - **Binning** —- Divide numerical scale into bins (think histogram) to then convert to a categorical feature.
- Categorical binning involved merging rare category levels.
- Attempts to make the model more robust and prevent overfitting at the cost of losing information.

Example

The Kaggle competition, Bike Sharing Demand Prediction, dealt with forecasting the rental demand based on historical usage patterns in relation with weather, time and other data. Smart feature engineering was instrumental[†] in securing a place in the top 5 percentile of the leaderboard, using:

Hour Bins: Categorise the hour feature with number of bins determined using a decision tree.

Temp Bins: Similarly, a binned feature for the temperature variable for both registered and casual users.

Year Bins: 8 quarterly bins were created for a period of 2 years, i.e., quarterly bins.

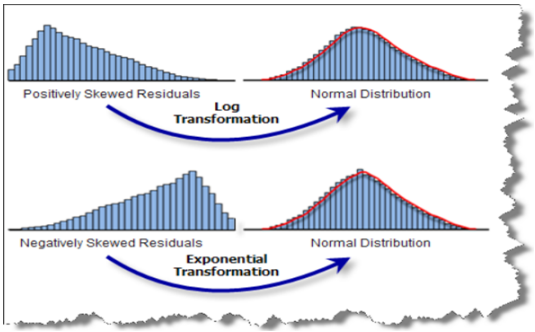
Day Type: Days were categorised as ‘weekday’, ‘weekend’ or ‘holiday’.

[†] Kaggle Bike Sharing Demand Prediction — How I got in top 5 percentile of participants?

Transform (typically Log/Power or similar)

Some machine learning models, like linear and logistic regression, have an assumption that variables follow a normal distribution. More likely, variables in datasets have skewed distribution. A log transformation can help to resolve this:

- Helps to handle skewed data and after transformation, the distribution becomes more approximate to normal
- Decreases the effect of the outliers due to the normalisation of magnitude differences and the model become more robust.
  (log function shrinks values bigger than one)
- The data you apply log transform to must have only positive values, otherwise you receive an error
  (We first add a value to make all values positive or bigger than one)



Categorical Encoding

- There are multiple techniques for encoding categorical variables[‡]
- One-hot encoding is one of the most common encoding methods in machine learning. This method spreads the values in a column to multiple flag columns and assigns 0 or 1 to them.

City
Rome
Madrid
Madrid
Istanbul
Istanbul
Istanbul
Rome

→

Istanbul	Madrid	Rome
0	0	1
0	1	0
0	1	0
1	0	0
1	0	0
1	0	0
0	0	1

- See sklearn.preprocessing for sklearn's list of encoders.

[‡]Python package, Category Encoder, has 15 encoders. Sklearn has 10.

Normalisation and Scaling

In most dataset, the numerical features of the dataset do not have a common range and/or common centre. Some models (for example PCA) are negatively affected by dominating variables.

Normalisation

- Normalisation (or min-max normalisation) scales all values in a fixed range, defaults between 0 and 1.
- This transformation does not change the distribution of the feature and due to the decreased standard deviations, the effects of the outliers increases. Therefore, before normalisation, it is recommended to handle the outliers

Standardisation

- Standardisation (or z-score normalisation) scales the values while taking into account standard deviation.
- If the standard deviation of features is different, their range also would differ from each other. This reduces the effect of the outliers in the features.

Feature Selection

Feature Selection

to efficiently find a minimum set of features that contain all the substantial information needed for predicting the target value.

- Initial feature vectors are typically not optimal since they may contain many redundant or irrelevant features.
- Hence their further processing is needed.
- Feature selection does not change existing features — it aims at selecting a subset of the most relevant features in order to both reduce the dimension of the feature vectors and remove unuseful 'noisy' initial features.
- Effect of feature selection is a more compact feature set:
  - improved model interpretability,
  - shorter training times,
  - enhanced prediction performance
    - lower dimension of feature vectors can reduce the risk of overfitting.
    - some learning methods do not work well if the feature set contains highly dependent features (e.g., Naïve Bayes learner, or SVM).

Feature selection methods — basic approaches

Feature selection methods can be divided into:

Filters

- Selects feature subsets as a pre-processing step, independent of the learning method.
- Since features are selected based on criteria independent of any supervised learner, the performance of filters may not be optimum for a chosen learner.
- Typical tests: Correlation coefficients (Pearson's, $\phi - k$), Chi-square, Information Gain, Gini Index, etc.

Wrappers

- Use an ML algorithm in conjunction with internal cross validation to score feature subsets by measuring their predictive power.
- Use a learner as a black box to evaluate the relative usefulness of a feature subset.
- Wrappers search the best feature subset for a given supervised learner, however, wrappers tend to be computationally expensive.

Embedded methods

- Perform feature selection during the process of training.
- Instead of treating a learner as a black box, embedded methods select features using the information obtained from training a learner.

Example: Selecting Number of Features via PCA I

Returning to the Wisconsin breast cancer dataset, lets filter by taking only the first 2 PCA components ...

```
22 from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression

pipeline = Pipeline([
    ('scl', StandardScaler()),
    ('pca', PCA(n_components=2)),
    ('clf', LogisticRegression(solver='lbfgs'))
])

scores = cross_val_score(estimator=pipeline, X=X_train, y=y_train, cv=10, n_jobs=-1)

print('CV accuracy scores: %s' % scores)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

Example: Selecting Number of Features via PCA I

Returning to the Wisconsin breast cancer dataset, lets filter by taking only the first 2 PCA components ...

```
23 from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression

pipeline = Pipeline([
    ('scl', StandardScaler()),
    ('pca', PCA(n_components=2)),
    ('clf', LogisticRegression(solver='lbfgs'))
])

scores = cross_val_score(estimator=pipeline, X=X_train, y=y_train, cv=10, n_jobs=-1)

print('CV accuracy scores: %s' % scores)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores),np.std(scores)))
```

```
CV accuracy scores: [0.97142857 0.94117647 0.91176471 0.91176471 0.970
 0.94117647 0.91176471 0.97058824 0.91176471]
CV accuracy: 0.941 +/- 0.026
```

- CV score dropped from 97% to 94%, but model is now much simpler as now using only 2 features instead of 30.
- Perhaps we have dropped too many ... this is the role of hyper-parameter tuning.

Recap of where we are with the Wisconsin breast cancer dataset

- Clean dataset with no missing values, but have 30 numerical (continuous) features with reasonable expectation of multi-collinearity issues[§].
- Given the dimension of 30, it seems reasonable to apply PCA, but how many principal components should we pick?

`n_components`

- Given the suspected multi-collinearity, the regularisation in the logistic regression is important.
  - What type of penalty (L_1 vs L_2) should we use?

`penalty`

  - How important should be the penalty be?

`C`

We want a general procedure to determine optimal values of these hyperparameters:

- First approach is to generate **validation curves** which will look at each parameter in turn.
- Then we will look at more automatic techniques — grid and random searches.

Recap of where we are with the Wisconsin breast cancer dataset

- Clean dataset with no missing values, but have 30 numerical (continuous) features with reasonable expectation of multi-collinearity issues[§].
- Given the dimension of 30, it seems reasonable to apply PCA, but how many principal components should we pick?

  `n_components`
- Given the suspected multi-collinearity, the regularisation in the logistic regression is important.
  - What type of penalty (L_1 vs L_2) should we use?

    `penalty`
  - How important should be the penalty be?

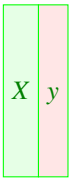
    `C`

We want a general procedure to determine optimal values of these hyperparameters:

- First approach is to generate **validation curves** which will look at each parameter in turn.
- Then we will look at more automatic techniques — grid and random searches.

Outline

The Problem

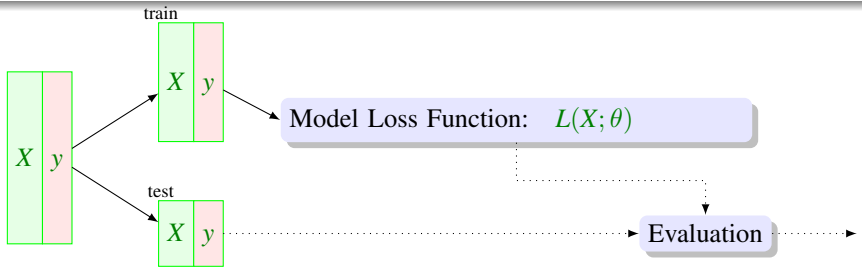


Model Loss Function: $L(X;\theta)$

- To date our model training has focused on optimising the model parameters, but a model can have additional parameters (called hyperparameters) whose values can have significant impact on the performance of the model.
- Usually cannot estimate the hyperparameters as part of the learning step because:
  - Hyperparameters play a quantifiable different role — think, degree of polynomial vs polynomial coefficients.
  - If not continuous — think L_1 vs L_2 option — then can't be used in gradient optimisation methods.
  - Greatly increase the complexity of the learning process.

$\Rightarrow$ Need a separate step to determine optimal values for the hyperparameters.

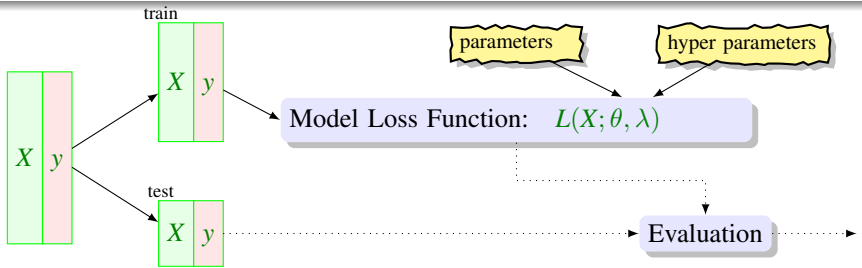
The Problem



- To date our model training has focused on optimising the model parameters, but a model can have additional parameters (called hyperparameters) whose values can have significant impact on the performance of the model.
- Usually cannot estimate the hyperparameters as part of the learning step because:
  - Hyperparameters play a quantifiable different role — think, degree of polynomial vs polynomial coefficients.
  - If not continuous — think L_1 vs L_2 option — then can't be used in gradient optimisation methods.
  - Greatly increase the complexity of the learning process.

$\Rightarrow$ Need a separate step to determine optimal values for the hyperparameters.

The Problem



- To date our model training has focused on optimising the model parameters, but a model can have additional parameters (called hyperparameters) whose values can have significant impact on the performance of the model.
- Usually cannot estimate the hyperparameters as part of the learning step because:
  - Hyperparameters play a quantifiable different role — think, degree of polynomial vs polynomial coefficients.
  - If not continuous — think L_1 vs L_2 option — then can't be used in gradient optimisation methods.
  - Greatly increase the complexity of the learning process.

$\Rightarrow$ Need a separate step to determine optimal values for the hyperparameters.

Parameters vs Hyperparameters

Parameters

- Weights/coefficients/numbers learnt during the training process.
- Automatically estimated.
- Examples:
  - coefficients in a linear / logistic regression.
  - support vectors in a support vector machine.
  - weights in an artificial neural network.

Hyperparameters

- 'Knobs'/'dials'/'switches' used to control the training process.
- Manually specified/set.
- Often used in processes to help estimate model parameters.
- Often set using heuristics.
- Examples:
  - Model selection
  - Feature selection
  - Size/depth of neural networks

Advances in machine learning techniques result in hyperparameters becoming parameters as algorithms improve and computational power increases.

Effect of PCA hyperparameter `n_components`

Returning to the Wisconsin breast cancer dataset, we want to see the effect that the number of components in the PCA step has on the overall model score.

```
1  from sklearn.pipeline import Pipeline
2  from sklearn.preprocessing import StandardScaler
3  from sklearn.decomposition import PCA
4  from sklearn.linear_model import LogisticRegression
5  pipeline = Pipeline([
6      ('scl', StandardScaler()),
7      ('pca', PCA(n_components=10)),
8      ('clf', LogisticRegression(solver='liblinear', penalty='l2'))
9  ])
10
11 from sklearn.model_selection import validation_curve
12 param_range = range(2,20)
13 train_scores, test_scores = validation_curve(
14     estimator=pipeline, X=X_train, y=y_train, cv=10,
15     param_name='pca__n_components', param_range=param_range)
```

Pipeline now has three steps: Scaling → PCA → Classifier

Parameter name `pca__n_components` is concatenation of pipeline step name, `pca`, and the parameter name `n_components`.

We specify a range (or set of values) for our hyper-parameter (line 11), then pass the pipeline and (hyper-)parameter info to the `validation_curve` function (line 13).

Effect of PCA hyperparameter `n_components` I

Returning to the Wisconsin breast cancer dataset, we want to see the effect that the number of components in the PCA step has on the overall model score.

```
25
1  from sklearn.pipeline import Pipeline
2  from sklearn.preprocessing import StandardScaler
3  from sklearn.decomposition import PCA
4  from sklearn.linear_model import LogisticRegression
5  pipeline = Pipeline([
6      ('scl', StandardScaler()),
7      ('pca', PCA(n_components=10)),
8      ('clf', LogisticRegression(solver='liblinear', penalty='l2'))
9  ])
10
11 from sklearn.model_selection import validation_curve
12 param_range = range(2,20)
13 train_scores, test_scores = validation_curve(
14     estimator=pipeline, X=X_train, y=y_train, cv=10,
15     param_name='pca__n_components', param_range=param_range)
```

Pipeline now has three steps:
Scaling → PCA → Classifier

Parameter name
`pca__n_components`
is concatenation of pipeline step name, `pca`, and the parameter name `n_components`.

We specify a range (or set of values) for our hyper-parameter (line 11), then pass the pipeline and (hyper-)parameter info to the `validation_curve` function (line 13).

Effect of PCA hyperparameter `n_components` II

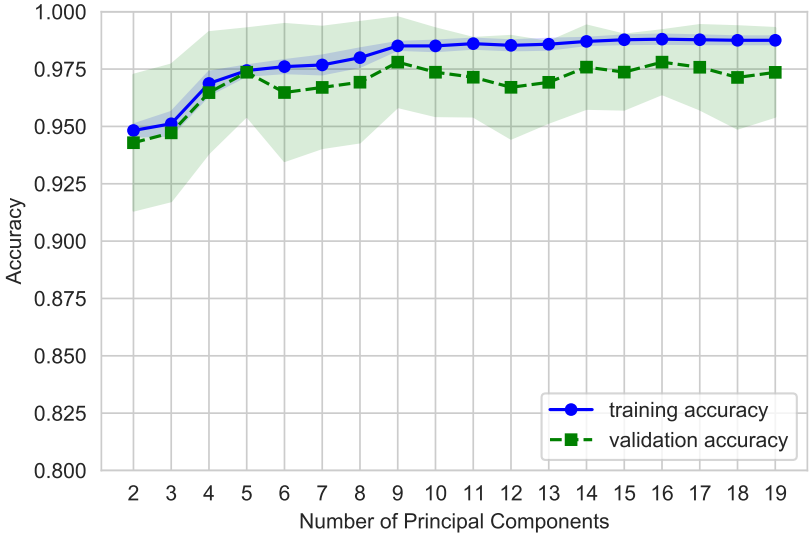
Next we compute statistics from the generated scores ...

```
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)
```

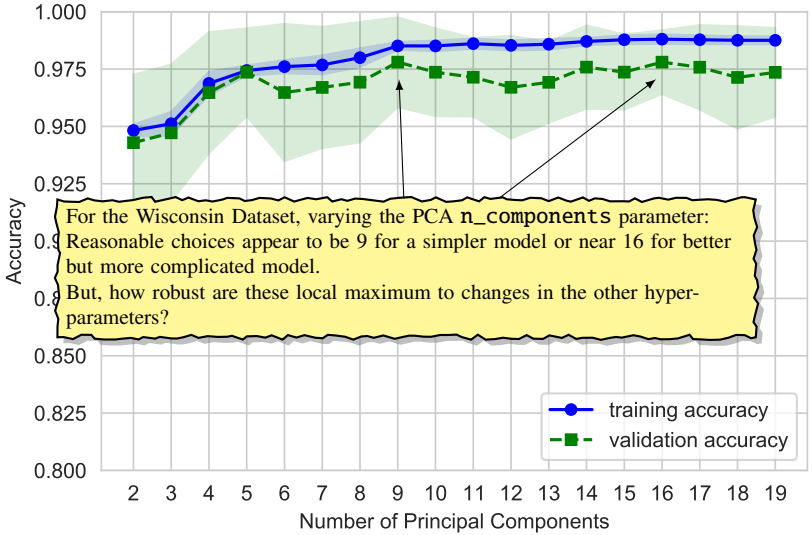
And then generate the actual validation curve using the usual plot and pimping code.

```
plt.plot(param_range, train_mean, color='blue', marker='o', markersize=5,
    label='training accuracy')
plt.fill_between(param_range, train_mean + train_std, train_mean - train_std,
    alpha=0.15, color='blue')

plt.plot(param_range, test_mean, color='green', marker='s', markersize=5,
    linestyle='--', label='validation accuracy')
plt.fill_between(param_range, test_mean + test_std, test_mean - test_std,
    alpha=0.15, color='green')

plt.xlabel('Number of Principal Components')
plt.ylabel('Accuracy')
plt.xticks(range(2,20))
plt.legend(loc='lower right')
plt.ylim(0.8, 1.0)
```

Effect of PCA hyperparameter `n_components` III



Effect of PCA hyperparameter `n_components` III



For the Wisconsin Dataset, varying the PCA `n_components` parameter: Reasonable choices appear to be 9 for a simpler model or near 16 for better but more complicated model.
But, how robust are these local maximum to changes in the other hyper-parameters?

The Validation Curve shows the sensitivity between model's accuracy with change in some (hyper-)parameter of the model.

- Two curves are present in a validation curve — one for the training set score and one for the cross-validation score.
- Ideally validation score and the training score look as similar as possible.

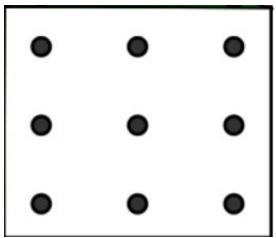
Hyper-parameter Tuning via Search

Validation curves are important but examining at each parameter individually is time consuming and does not take into account interactions between parameters — in effect it is a series of 1D semi-manual searches.

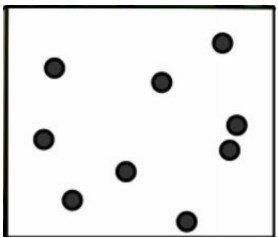
Parameter Search

Construct a parameter space (list of all possible hyper-parameter value combinations) and search them, and choose the one with the best cross validation results.

Grid Search



Random Search



Grid Search (`sklearn.model_selection.GridSearchCV`)

Grid Search

Try all available parameters combinations, 1 by 1, and choose the one with the best cross validation results.

Drawbacks

- Still very slow — trying ALL combinations of ALL parameters with no modification of search based on results found to date.
- ALL combinations $\Longrightarrow$ every additional hyperparameter to vary multiplies the number of iterations you need to complete.
- It can work only with discrete values.
  If the global optimum is on `n_estimators=550`, but you are doing `GridSearchCV` from 100 to 1000 with step 100, you will never reach the optimal point.

Stratagies

- You need know / guess the approximate region of the optimum to start.
- To mitigate against drawbacks, do multiple lower dimensional grid searches, or repeat search with narrower grids and smaller step sizes.

Grid Search Example


```
28 from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
pipeline = Pipeline([
    ('scl', StandardScaler()),
    ('clf', SVC(random_state=SEED))])

param_range = np.logspace(-4,4,9)
param_grid = [{
    'clf__gamma': param_range,
    'clf__C': param_range,
    'clf__kernel': ['rbf']
}]
gs = GridSearchCV(estimator=pipeline,
    return_train_score=True,
    param_grid=param_grid, scoring='accuracy', cv=10, n_jobs=-1)
```

- I switched from LogisticRegression to Support Vector machine (SVM) — more hyperparameters to tweak.
- How many parameter combinations were generated/used?

Grid Search Example II

To perform grid search, we call the `fit` method as usual . . .

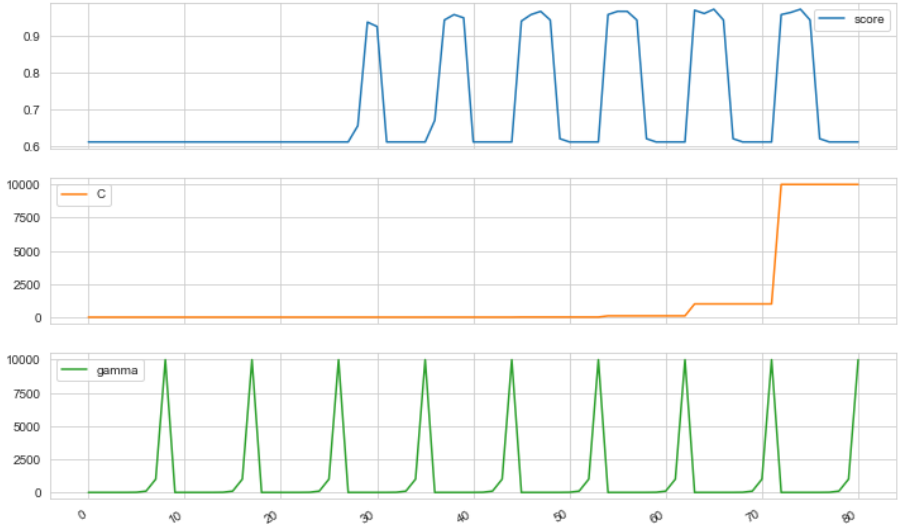
29
```
gs = gs.fit(X_train, y_train)
print(gs.best_score_)
print(gs.best_params_)
```

```
0.9735294117647058
{'clf__C': 1000.0, 'clf__gamma': 0.01, 'clf__kernel': 'rbf'}
```

- CV score is reported with optimal values of the hyper-parameters.
- `GridSearchCV` with option `return_train_score=True`, returns results of each combination — determine effect of hyperparameters to refine grid search.

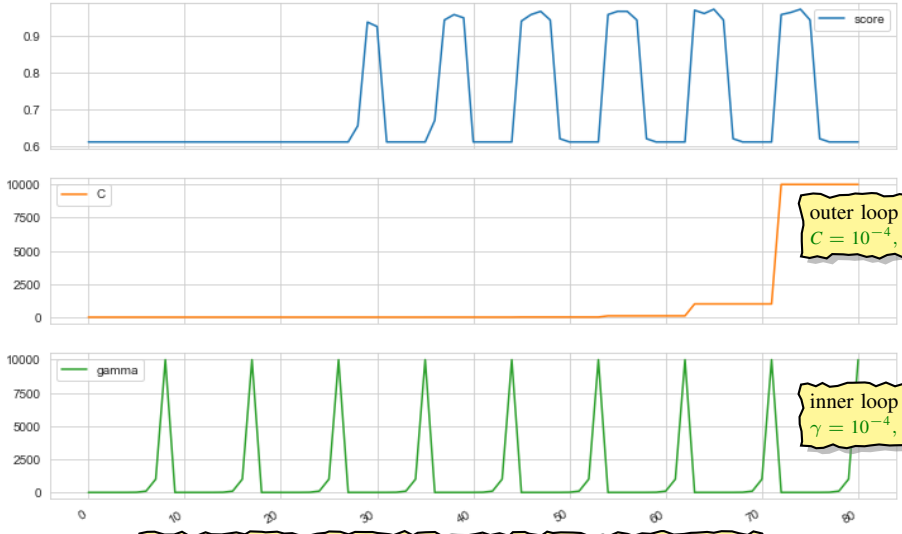
30
```
df_gs = pd.DataFrame(np.transpose([
    gs.cv_results_["mean_test_score"],
    gs.cv_results_["param_clf__C"].data,
    gs.cv_results_["param_clf__gamma"].data]),
    columns=['score', 'C', 'gamma'])

df_gs.plot(subplots=True,figsize=(12, 8))
plt.savefig("gs__svm__C_gamma.pdf", bbox_inches="tight")
plt.show()
```

Grid Search Example III



Horizontal axis is point in hyper-parameter search space index

Grid Search Example

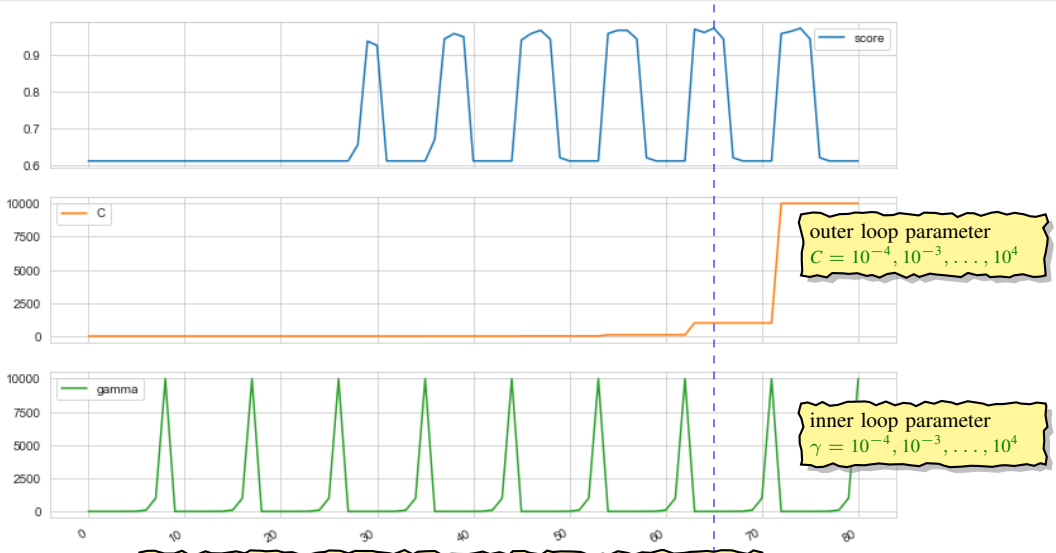


outer loop parameter
$C = 10^{-4}, 10^{-3}, \ldots, 10^{4}$

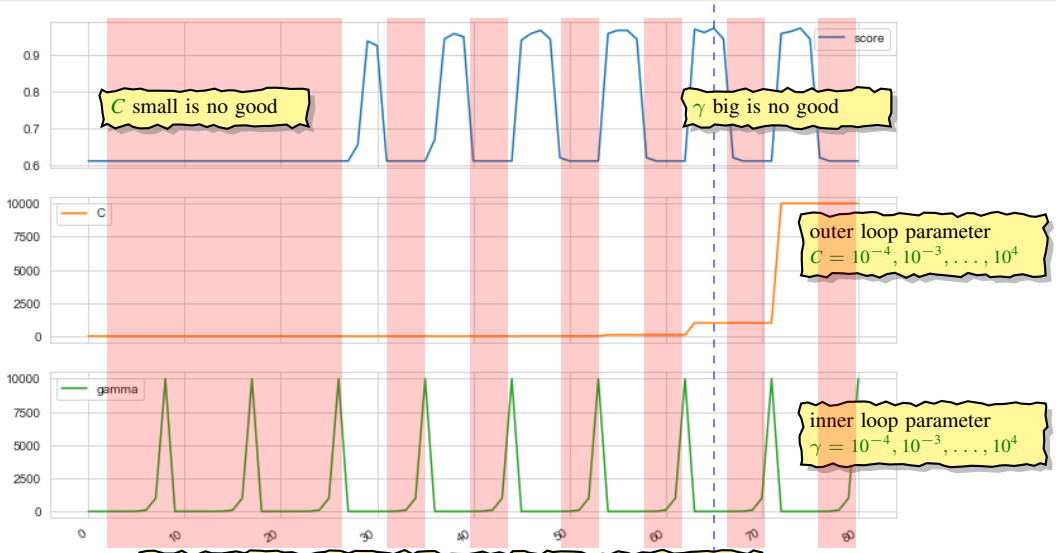
inner loop parameter
$\gamma = 10^{-4}, 10^{-3}, \ldots, 10^{4}$

Horizontal axis is point in hyper-parameter search space index

Grid Search Example ($C = 1000, \gamma = 0.01$) III



Grid Search Example ($C = 1000, \gamma = 0.01$) III

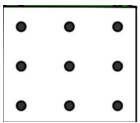


Random Search

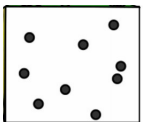
Random Search

Try random hyperparameter combinations and choose the one with the best cross validation results.

Grid Search



Random Search



Advantages

- On every step random search varies all parameters $\Rightarrow$ doesn't spend time on meaningless parameters.
- On average finds near optimal parameters much faster than Grid search.
- It is not limited by grid when optimising continuous parameters.

Disadvantages

- It may not find the global optimal parameter on a grid.
- All steps are independent. Does not use information about the results gathered to inform search.

Random Search


```
31
1  pipeline = Pipeline([
2      ('scl', StandardScaler()),
3      ('clf', SVC(kernel='rbf', random_state=SEED))
4  ])
5
6  from sklearn.model_selection import RandomizedSearchCV
7  from scipy.stats import randint
8
9  param_range = np.logspace(-4,4,40)
10
11 param_grid = {'clf__C': param_range,
12                'clf__gamma': param_range}
13
14 rs = RandomizedSearchCV(estimator=pipeline,
15     param_distributions=param_grid,
16     n_iter = 81, random_state=SEED,
17     return_train_score=True,
18     scoring='accuracy', cv=10, n_jobs=-1)
```

- This is the same setup as used in GridSearch but the number of parameter values combinations went from 81 to 1600, but still only sampling 81 points.

Random Search Example II

To perform grid search, we call the `fit` method as usual ...

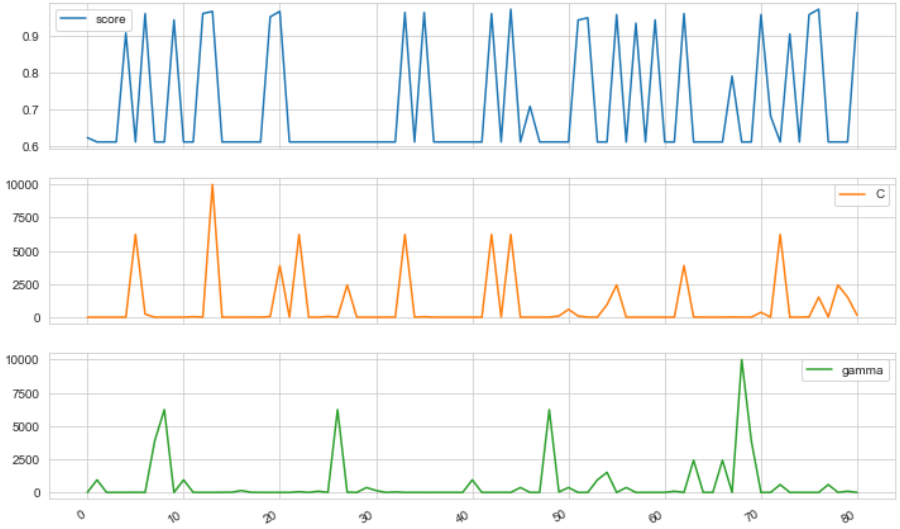
32
```
rs.fit(X_train, y_train)
print(rs.best_score_)
print(rs.best_params_)
```

```
0.9736134453781513
{'clf__gamma': 0.0004124626382901352, 'clf__C': 6235.507341273912}
```

- Sampling the same number of points (81) in a search space nearly 20 times larger, RandomSearch outperforms GridSearch.

33
```
df_rs = pd.DataFrame(np.transpose([
    rs.cv_results_["mean_test_score"],
    rs.cv_results_["param_clf__C"].data,
    rs.cv_results_["param_clf__gamma"].data]),
    columns=['score', 'C', 'gamma'])

df_rs.plot(subplots=True,figsize=(12, 8))
plt.savefig("rs__example_1.pdf", bbox_inches="tight")
plt.show()
```

Random Search Example III



Every step is completely random. $\Rightarrow$ does not spent time on useless parameters, but does not use the information gathered on the first steps to improve outcomes of the latter ones.

The Problem with GridSearch and RandomSearch

Both GridSearch and RandomSearch ignore information gained during the search, by using such information can searches improve?

- What information would be useful?
  - Function value
  - Function derivative (gradient) — gives direction in which the function is decreasing (locally).
  - Making decisions based on local information could locate global minimum.
- What can go wrong?
  - Multiple local minimum — can get stuck in a side valley.
    - There will be situations in which we should make decisions against current function value information — see Simulated Annealing.
  - Gradient could be flat (so no direction to go in), or worse almost flat (some algorithms become unstable), or non-existent.
- So what is the solution?

Sequential model-based optimisation (SMBO)
(also known as Bayesian optimisation) is a general technique for function optimisation that uses information from earlier function evaluations to refine the optimisation search — with the aim of minimising number of function calls.

Outline

Motivation for Ensemble Learnering

Condorcet's jury theorem is a political science theorem about the relative probability of a given group of individuals arriving at a correct decision:

Theorem 1 (Condorcet's jury theorem)

Assume a group of n independent voters wishes to reach a decision by majority vote. One of the two outcomes of the vote is correct, and each voter has an independent probability, p, of voting for the correct decision.

- *If $p > 0.5$, then adding more voters increases the probability that the majority decision is correct. In the limit, the probability that the majority votes correctly approaches 1 as the number of voters, n increases.*
- *If $p < 0.5$ then adding more voters makes things worse: the optimal jury consists of a single voter.*

Take Home Message

- Even weak decision makers have benefit as long as they individualy perform better than chance ($p > 0.5$).
- "Wisdom of Crowds" needs independence!
- What happens if $p < 0.5$?

Practical Motivation — Netflix Prize

- Open competition* to predict user ratings for films, based only on previous ratings.
- Prize was awarded on 2009 to BellKor's Pragmatic Chaos team which bested Netflix's own algorithm for predicting ratings by 10.06%, using an ensemble of 107 modules.

Leaderboard

Display top 20 leaders.

Rank	Team Name	Best Score	% Improvement	Last Submit Time
1	The Ensemble	0.8553	10.10	2009-07-26 18:38:22
2	BellKor's Pragmatic Chaos	0.8554	10.09	2009-07-26 18:18:28
Grand Prize - RMSE <= 0.8563				
3	Grand Prize Team	0.8571	9.91	2009-07-24 13:07:49
4	Opera Solutions and Vandelay United	0.8573	9.89	2009-07-25 20:05:52
5	Vandelay Industries !	0.8579	9.83	2009-07-26 02:49:53
6	PragmaticTheory	0.8582	9.80	2009-07-12 15:09:53
7	BellKor in BigChaos	0.8590	9.71	2009-07-26 12:57:25
8	Dace	0.8603	9.58	2009-07-24 17:18:43
9	Opera Solutions	0.8611	9.49	2009-07-26 18:02:08
10	BellKor	0.8612	9.48	2009-07-26 17:19:11
11	BigChaos	0.8613	9.47	2009-06-23 23:06:52
12	Feeds2	0.8613	9.47	2009-07-24 20:06:46
Progress Prize 2008 - RMSE = 0.8616 - Winning Team: BellKor in BigChaos				
13	xiangliang	0.8633	9.25	2009-07-21 02:04:40
14	Gravity	0.8634	9.25	2009-07-26 15:58:34
15	Ces	0.8642	9.17	2009-07-25 17:42:38
16	Invisible Ideas	0.8644	9.14	2009-07-20 03:26:12
17	Just a guy in a garage	0.8650	9.08	2009-07-22 14:10:42
18	Craig Carmichael	0.8656	9.02	2009-07-25 16:00:54
19	J Dennis Su	0.8658	9.00	2009-03-11 09:41:54
20	acmehill	0.8659	8.99	2009-04-16 06:29:35
Progress Prize 2007 - RMSE = 0.8712 - Winning Team: KorBell				
Cinematch score on quiz subset - RMSE = 0.9514				

*Netflix Prize

Ensembe Methods

Definition 2 (Ensemble Learner)

An ensemble learner is a set of models whose individual decisions are combined in some way to classify new examples.

- Simplest approach:
  - Generate multiple classifiers
  - Each votes on test instance
  - Take majority as classification
- Classifiers are different due to different sampling of training data, or randomised parameters within the classification algorithm.
- Aim: take simple mediocre algorithm and transform it into a super classifier without requiring any fancy new algorithm.
- Differ in training strategy, and combination method:
  - Parallel training with different training sets: Bagging or Cross-validated committees
  - Sequential training, iteratively re-weighting training examples so current classifier focuses on hard examples: boosting
  - Parallel training with objective encouraging division of labor: mixture of experts

Why do Ensemble Methods Work?

Variance reduction

If the training sets are completely independent, it will always help to average an ensemble because this will reduce variance without affecting bias (e.g., bagging)

- Reduce sensitivity to individual data points.

Bias reduction

For simple models, average of models has much greater capacity than single model (e.g., hyperplane classifiers, Gaussian densities).

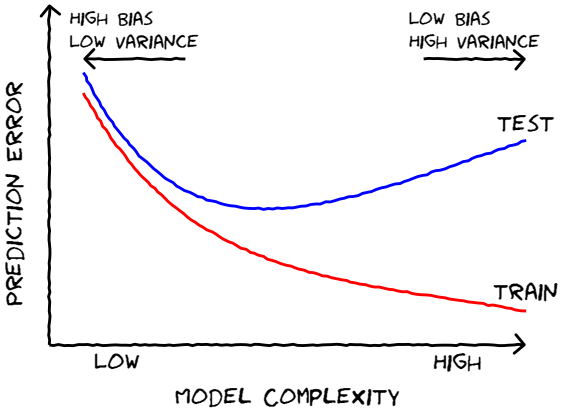
- Averaging models can reduce bias substantially by increasing capacity, and control variance by fitting one component at a time (e.g., boosting)

How de we combine models — classification vs regression?

- Regression models can be averaged.
- Classification models can be averaged if output is probability of being in a class, otherwise can use majority vote if classifier only outputs class.

Bias/Variance Tradeoff

Bias: error from erroneous assumptions in the model



Variance: error from sensitivity to small fluctuations in the training set

- Model too simple then has large bias (as model is too simple to learn signal) but small variance (as model is too simple to learn/be affected by noise).
- Model too complicated then has small bias (as model can learn signal) but has large variance (as model also can learn noise).

Reduce Variance Without Increasing Bias

It seems that all we can do to is select how complicated our model is to minimise the generalisation error ($\text{bias}^2 + \text{Var} + \text{noise}$). But this is not the case:

- It is possible to reduce variance without affecting bias by averaging.

Averaging reduces variance

- Given N independent estimates for X, each with variance of $\text{Var}(X)$, we have

$$\text{Var}(\bar{X}) = \frac{\text{Var}(X)}{N}$$

But only if independent!

One Problem

We have only one training set, so where do multiple models (independent estimates) come from? — apply Bootstrap sampling

Bootstrap sampling

Given set D containing N training examples, create D' by drawing N examples at random with replacement from D.

Bagging aka Bootstrap AGgregation

Method

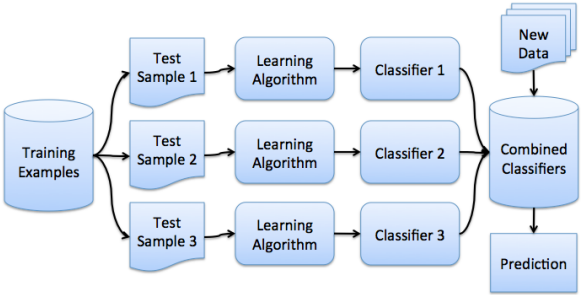
- Create M bootstrap samples, $D_1, D_2, \ldots, D_M$ of size N.
  - When picking each element of sample D_i, each element in D has probability of $1/N$ of being selected.
  - The probability of an element of D not being selected for D_i is

$$(1 - 1/N)^N \xrightarrow[N \to \infty]{} 1/e$$

  - Hence the probability of an element of D being selected for D_i is $1 - 1/e = 0.632$.

A bootstrap sample contains 63% of the original data.

- Separately train classifier on each D_i.
- Classify new instance by majority vote / average.



Bagging aka Bootstrap AGgregation

Method

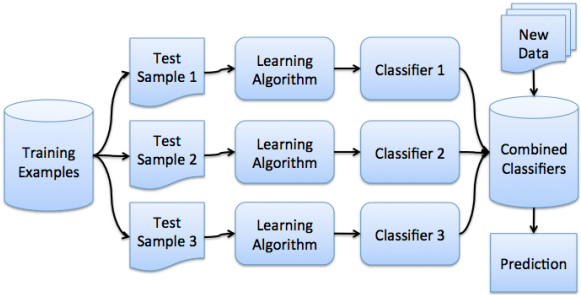
- Create M bootstrap samples, $D_1, D_2, \ldots, D_M$ of size N.
  - When picking each element of sample D_i, each element in D has probability of $1/N$ of being selected.
  - The probability of an element of D not being selected for D_i is

$$(1 - 1/N)^N \xrightarrow[N \to \infty]{} 1/e$$

  - Hence the probability of an element of D being selected for D_i is $1 - 1/e = 0.632$.

A bootstrap sample contains 63% of the original data.

- Separately train classifier on each D_i.
- Classify new instance by majority vote / average.



Bagging — Performance

Definition 3 (Unstable learner)

A learner is unstable if its output classifier undergoes major changes in response to small changes in training data

- Unstable: decision-tree, neural networks, rule learning algorithms, ...
- Stable: linear regression, logistic regression, nearest neighbour, linear threshold algorithms, ...

Bagging tends to

- works well for unstable learners
- can have a mild negative effect on the performance of stable methods

Best Case

$$\mathrm{Var}\Big(\mathrm{Bagging}\big(L(x,D)\big)\Big) = \frac{\mathrm{Var}\big(L(x,D)\big)}{M}$$

However, in practice the models are correlated, so reduction is smaller than $1/M$. Also variance of models trained on fewer training cases can be somewhat larger.

Boosting — Motivation — ‘How May I Help You?’

Problem

Automatically categorise type of call requested by phone customer (`Collect`, `CallingCard`, `PersonToPerson`, etc.)

- ‘Yes I’d like to place a collect call long distance please’ (`Collect`)
- ‘Operator I need to make a call but I need to bill it to my office’ (`ThirdNumber`)
- ‘Yes I’d like to place a call on my master card please’ (`CallingCard`)
- ‘I just called a number in sioux city and I musta rang the wrong number because I got the wrong party and I would like to have that taken off of my bill’ (`BillingCredit`)

Observations

- Easy to find ‘rules of thumb’ that are ‘often’ correct
  e.g. ‘IF "card" occurs in utterance THEN predict ‘`CallingCard`’
- Hard to find single highly accurate prediction rule.

Boosting Approach

Outline

- Devise procedure for deriving rough rules of thumb
- Apply procedure to subset of examples
- Obtain rule of thumb
- Apply to 2nd subset of examples
- Obtain 2nd rule of thumb . . . and repeat till done.

Key Steps

- How do we choose examples on each round?
  - Concentrate on hardest examples (those most often miss-classified by previous rules of thumb) — focus by sampling or weighing the whole data set.
- How do we combine rules of thumb into a single prediction rule?
  - Take (weighted) majority vote of rules of thumb.

If Boosting is possible, then

- can use (fairly) wild guesses to produce highly accurate predictions.
- for any learning problem:
  - either can always learn with nearly perfect accuracy
  - or there exist cases where cannot learn even slightly better than random guessing

Boosting

Boosting
General method of converting rough rules of thumb into highly accurate prediction rule.

- Assume given 'weak' learning algorithm that can consistently find classifiers ('rules of thumb') at least slightly better than random, say, accuracy $\geq 55\%$ (in two-class setting).

'weak learning assumption'

- Then given sufficient data, a boosting algorithm can provably construct single classifier with very high accuracy, say, 99%.
- In contrast to bagging which has little effect on Bias, boosting reduces both bias and variance.

History

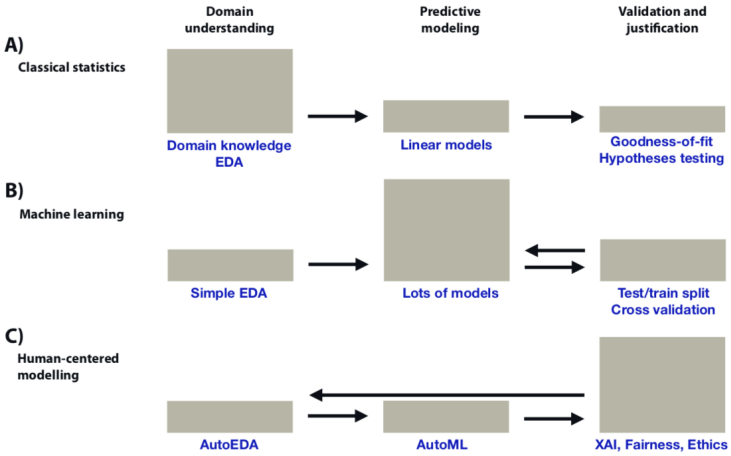
Schapire '89 — first provable boosting algorithm

Freund '90 — 'optimal' algorithm that 'boosts by majority'

nd & Schapire '95 — introduced 'AdaBoost' algorithm, strong practical advantages over previous boosting algorithms

Outline

Shift in Relative Importance in Data Science Modelling



(A→B)

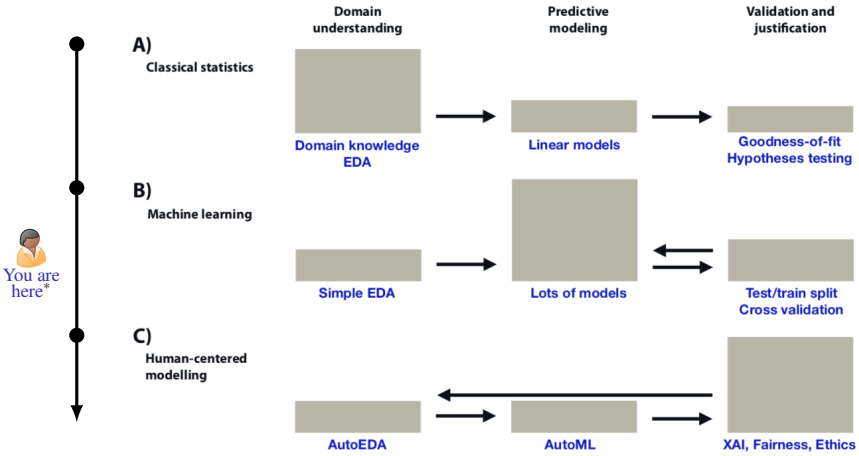
- Move from statistical modelling to machine learning (week 4).

(B→C)

- Increased automation of the hyper-parameter tuning, model fitting, model selection, feature engineering, EDA stages.
- Resulting models are better (score higher) but are more opaque.
- Increased need for model generic tools to explain/justify predictions

*Explanatory Model Analysis, pbiecek.github.io/ema/

Shift in Relative Importance in Data Science Modelling



(A→B)

- Move from statistical modelling to machine learning (week 4).

(B→C)

- Increased automation of the hyper-parameter tuning, model fitting, model selection, feature engineering, EDA stages.
- Resulting models are better (score higher) but are more opaque.
- Increased need for model generic tools to explain/justify predictions

*Explanatory Model Analysis, pbiecek.github.io/ema/

Why Should We Care About XAI?

> "With great power there must also come great responsibility"
>
> — Peter Parker principle, Marvel Comics

- Predictive models are only as good as the data they are trained on:
  - Google's image object identification was trained on a dataset of predominantly white people — resulting in classifying some black people as gorillias (WSJ, July 2015)
  - Nikon S630, Hewlett-Packard's MediaSmart web camera, ...
- Mass adoption of (the more "effective") models means that pre-existing systematic biases are propagated and become more baked in.
  - Amazon's same-day delivery service was unavailable for ZIP codes in predominantly black neighbourhoods — correlated to areas affected by mortgage redlining in the 1960s. (Bloomberg, 2016)
  - Amazon developed a internal recruiting recommendation engine but was initially trained on male dominated data and even after subsequence attempts to remove gender the model learnt to use proxies for gender (membership of women's tennis etc, or 'male oriented' verbs). (Reuters, 2019)
- Decisions made by models are harder to refute (Think Little Britain's "Computer says 'NO' ")
  - Software that assessed the risk of recidivism in criminals was twice as likely to mistakenly flag black defendants as being at a higher risk of committing future crimes. (ProPublica, 2016)

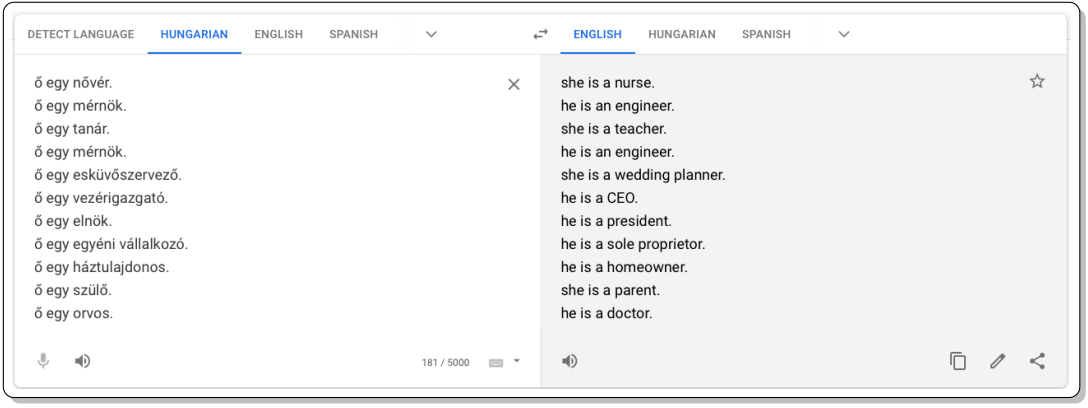
Example: Assuming Gender Based on Stereotypes

What do[†] nurses, teachers, wedding planners, or parents have, that engineers, CEOs, presidents, sole proprietors, home owners, and doctors don't?

[†]Full disclosure — I knew about nurse, engineer, wedding planner and CEO before. The other professions were my first picks. The only surprise is shop assistant. (March 8, 2021).

Example: Assuming Gender Based on Stereotypes

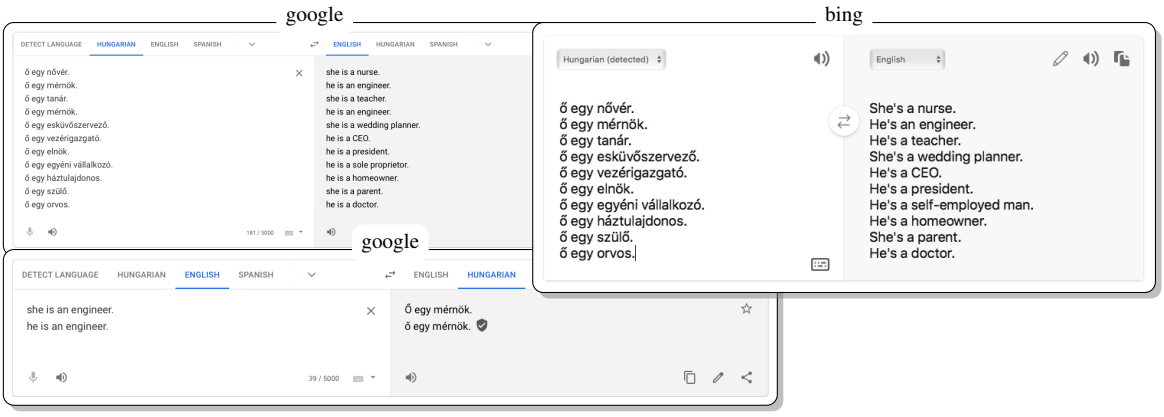
What do[†] nurses, teachers, wedding planners, or parents have, that engineers, CEOs, presidents, sole proprietors, home owners, and doctors don't?



[†]Full disclosure — I knew about nurse, engineer, wedding planner and CEO before. The other professions were my first picks. The only surprise is shop assistant. (March 8, 2021).

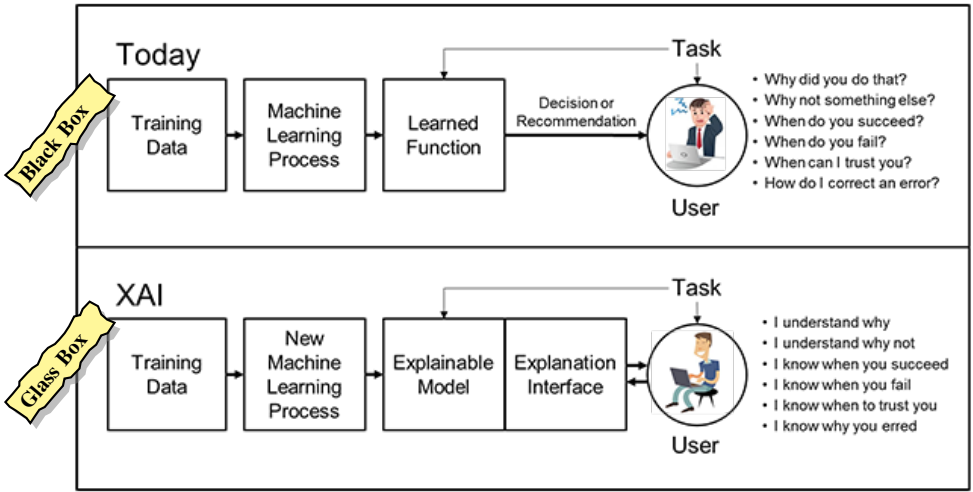
Example: Assuming Gender Based on Stereotypes

What do[†] nurses, teachers, wedding planners, or parents have, that engineers, CEOs, presidents, sole proprietors, home owners, and doctors don't? . . . and it is not just Google



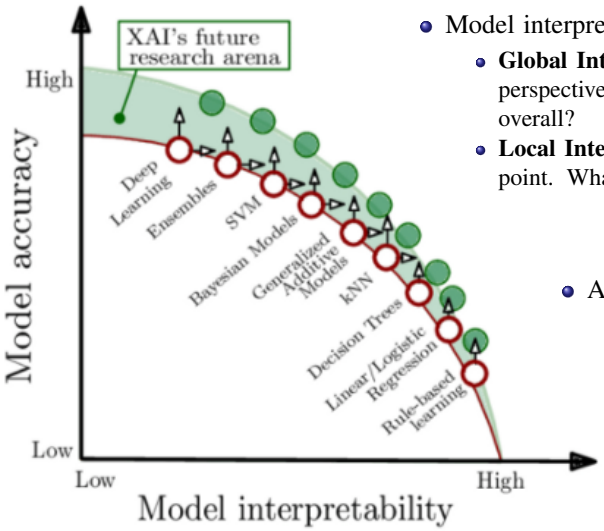
[†]Full disclosure — I knew about nurse, engineer, wedding planner and CEO before. The other professions were my first picks. The only surprise is shop assistant. (March 8, 2021).

The need for XAI‡



‡DARPA, Our Research, XAI, Dr. Matt Turek

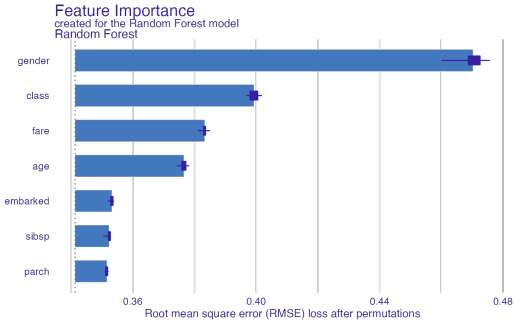
Accuracy vs Interpretability Trade Off

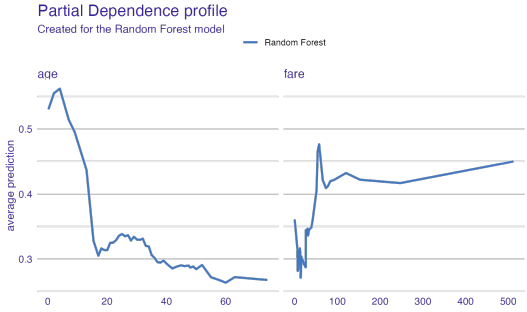


- Model interpretability can be examined in two levels:
  - **Global Interpretation** examines the model from an over all perspective. Which features are important and how important overall?
  - **Local Interpretation** is focused on a individual observation/data point. What features contributed to this prediction?

- Active area of research and development:
  - **LIME**
    Local Interpretable Model-Agnostic Explanations
  - **SHAP**
    Shapley Additive Explanations
  - **DALEX**
    moDel Agnostic Language for Exploration and eXplanation
  - ...

XAI Example: Titanic (via DALEX) — Global Interpretation

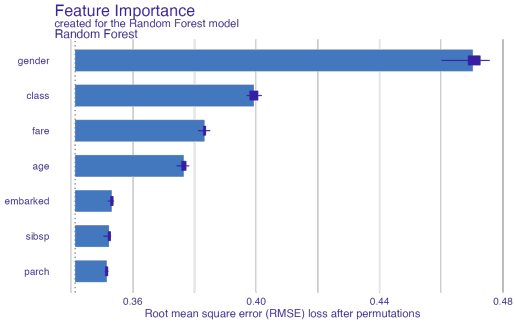


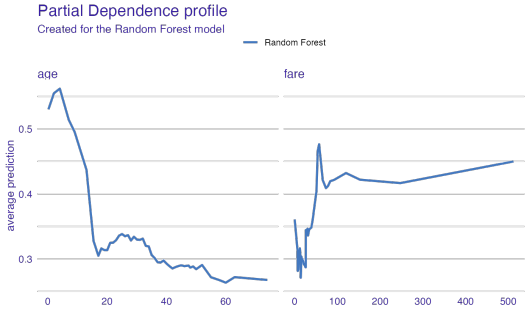


- The important feature is gender.
- Next three important features are class (1,2,3), age and fare.

- Kids under 5 have much higher probability of survival, drops significantly near 20.
- Fare reflects passenger class, why the spike?

XAI Example: Titanic (via DALEX) — Global Interpretation





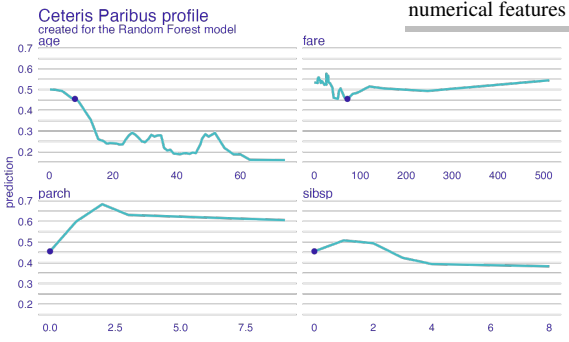
- The important feature is gender.
- Next three important features are class (1,2,3), age and fare.

- Kids under 5 have much higher probability of survival, drops significantly near 20.
- Fare reflects passenger class, why the spike?

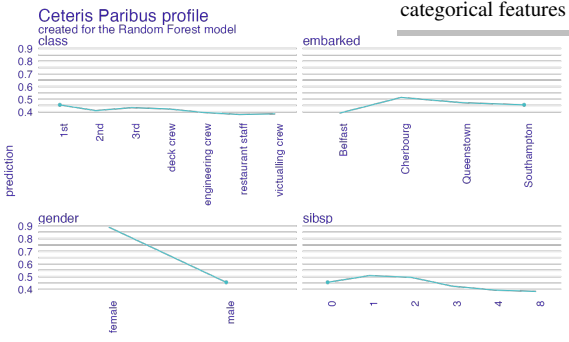
XAI Example: Titanic (via DALEX) — Local Interpretation

Lets break down explanation for model predictions for a fictitious, Ceteris Paribus, an 8 years old male, in 1st class that embarked from port C . . .

numerical features



categorical features

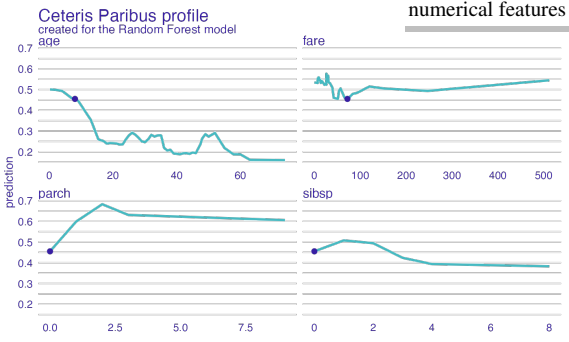


- It looks like the most important feature for this passenger is age and sex.
- His odds for survival are higher than for the average passenger. Mainly because of the young age and class, despite being a male.

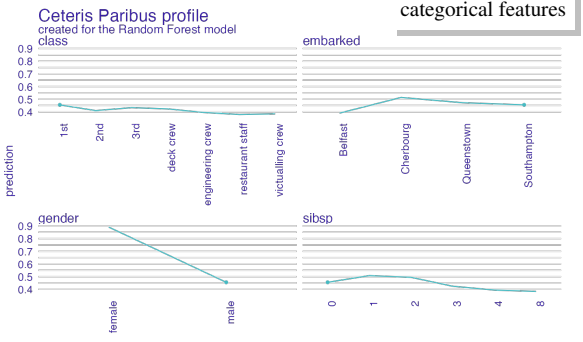
XAI Example: Titanic (via DALEX) — Local Interpretation

Lets break down explanation for model predictions for a fictitious, Ceteris Paribus, an 8 years old male, in 1st class that embarked from port C . . .

numerical features



categorical features



- It looks like the most important feature for this passenger is age and sex.
- His odds for survival are higher than for the average passenger. Mainly because of the young age and class, despite being a male.