» Evaluating Performance

- Machine learning system is being built for a business purpose e.g. create a new service, improve an existing service, increase profits, save time $\rightarrow$ important to be clear about this overall purpose
- How to measure how well overall objectives are met by a proposed ML system?
  - Probably multiple metrics are of interest, not just one
  - Goodhart's Law: *when a measure becomes a target it ceases to be a good measure* e.g. suppose use waiting time as metric for hospital emergency treatment - can be reduced by keeping patients in ambulance, treat people quickly and tell them to come back, favour easy cases
    $\rightarrow$ keep a focus on what outcomes really matter, rarely just minimising/maximising some metric(s)
  - Establish a baseline - how well do existing solutions perform, what are comparable problems, and how have they been solved?

» Choice of Metric For Regression

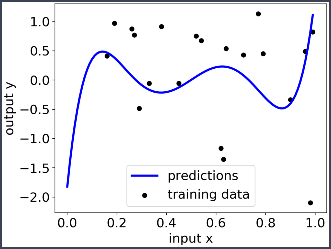
For regression problems we used the mean square error for model tuning using cross-validation. That or the RMSE are usually fine, but other choices are possible e.g.

* Mean square error $\frac{1}{m}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})^2$
* Root mean square error (RMSE) $\sqrt{\frac{1}{m}\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})^2}$
* Mean absolute error $\frac{1}{m}\sum_{i=1}^{m}|\theta^T x^{(i)} - y^{(i)}|$
  * Gives less weight to large errors than mean square error e.g if $\theta^T x^{(i)} - y^{(i)} = 100$ then $|\theta^T x^{(i)} - y^{(i)}| = 100$ while $(\theta^T x^{(i)} - y^{(i)})^2 = 100^2 = 10,000$.
* R^2: $1 - \frac{\sum_{i=1}^{m}(\theta^T x^{(i)} - y^{(i)})^2}{\sum_{i=1}^{m}(\theta^T x^{(i)} - \bar{y})^2}$ where $\bar{y} = \frac{1}{m}\sum_{i=1}^{m} y^{(i)}$ is the mean training data output.
  * The reduction in square error relative to just predicting a constant output.
  * $R^2 = 1$ when model predicts perfectly, and $R^2 = 0$ when prediction is no better than predicting the mean value.

» Comparison With A Baseline Predictor

How do you know whether the prediction error you've calculated on unseen test data is "good"? Example:

- Generate data where the output is just gaussian noise.
- Split this into training and test data, fit a linear regression model to the training data using polynomial features.
- Typical mean square error: 0.965738. Typical predictions:



- Have we achieved anything non-trivial? Data is just noise after all, so surely not. So how can we check?

» Comparison With A Baseline Predictor

* Compare the performance of our predictions against the performance of a trivial baseline estimator.
    * E.g. an estimator that always uses the mean value of the training data as its prediction i.e. prediction is a constant that doesn't depend on the input features at all.
    * sklearn.dummy.DummyRegressor and DummyClassifier
* For our gaussian noise example:
    * Mean square error of regression model: 0.965738
    * Mean square error of constant model: 0.787940
    * Trivial constant model has *lower* error than our polynomial regression model!
* You should always compare the quality of any predictions with a simple baseline model
    * This is mandatory in your projects and assignments
* Constant baseline model is a reasonable choice, but insight into problem at hand may suggest other baseline models worth considering. E.g. weather stats tell us that in Ireland predicting tomorrow's weather will be the same as today's will be right about 60% of the time, so that's a baseline to beat.

» Python Code For Baseline Predictor Gaussian Noise Example

```
import numpy as np
import numpy as np
X = np.arange(0,1,0.01).reshape(−1, 1)
y = np.random.normal(0.0,1.0,X.size).reshape(−1, 1)

from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X,y,test_size=0.2)

from sklearn.preprocessing import PolynomialFeatures
Xtrain_poly = PolynomialFeatures(6).fit_transform(Xtrain)
Xtest_poly = PolynomialFeatures(6).fit_transform(Xtest)
X_poly = PolynomialFeatures(6).fit_transform(X)

from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(Xtrain_poly, ytrain)

ypred = model.predict(Xtest_poly)

from sklearn.dummy import DummyRegressor
dummy = DummyRegressor(strategy=”mean”).fit(Xtrain_poly, ytrain)
ydummy = dummy.predict(Xtest_poly)

from sklearn.metrics import mean_squared_error
print(”square error %f %f”%(mean_squared_error(ytest,ypred),mean_squared_error(ytest,ydummy)))

import matplotlib.pyplot as plt
plt.rc(’font’, size=18); plt.rcParams[’figure.constrained_layout.use’] = True
plt.scatter(Xtest, ytest, color=’black’)
ypred = model.predict(X_poly)
plt.plot(X, ypred, color=’blue’, linewidth=3)
plt.xlabel(”input x”); plt.ylabel(”output y”)
plt.legend([”predictions”,”training data”])
plt.show()
```

» Choice of Metric For Classification

Suppose our task is to predict whether label associated with feature vector x is $+1$ or -1. Four possible outcomes:

* Prediction is correct
  * *True positive* we predict label $+1$ and correct label is $+1$
  * *True negative* we predict label -1 and correct label is -1
* Prediction is wrong
  * *False positive* we predict label $+1$ and correct label is -1 (also called a *false alarm*)
  * *False negative* we predict label -1 and correct label is $+1$
* E.g. in Covid app to restrict infection spread we want few false negatives but to minimise disruption we also want few false positives.
* There is a trade-off between true positives and false negatives.
  * E.g. if always predict $+1$ then we'll catch all positive cases (100% of positive data will be predicted as positive) but also create many false negatives (all negative data will also be predicted as positive).

» Imbalanced Data

* Why not just measure *accuracy* i.e. $\frac{\#\textit{correct predictions}}{\#\textit{total instances}}$?
* Suppose many more negative cases than positive cases (or vice versa) $\rightarrow$ *imbalanced data*. Imbalanced data is v common. E.g. most people are not infected with Covid.
* Suppose have 1000 data points and 1 is positive, rest are negative. Try a dummy classifier that always predicts -1. Its accuracy will be $999/1000$ or 99.9% !
* *Remember:* you should *always* compare the quality of any predictions with a simple baseline model, both when doing regression and classification - mandatory in your projects and assignments

» Confusion Matrix

true positive	TP	FN
true negative	FP	TN
	predicted positive	predicted negative

In previous example:

true positive	0	1
true negative	0	999
	predicted positive	predicted negative

with $m = 1000$ data points.

- *Accuracy* = $\frac{TN+TP}{TN+TP+FN+FP} = \frac{999}{1000}$
- *True positive rate* = $\frac{TP}{TP+FN} = \frac{0}{1} = 0$ (or *Recall*)
- *False positive rate* = $\frac{FP}{TN+FP} = \frac{0}{1} = 0$ (or *Specificity*)
- *Precision* = $\frac{TP}{TP+FP}$ (fraction of positive predictions which are correct)

» Movie Review Example

SVM with L_2 penalty parameter $C = 1.0$:

true positive	60	13
true negative	24	63
	predicted positive	predicted negative

with $m = 160$ data points (20% test split from full data set of 800 points).

- *Accuracy* = $\frac{TN+TP}{TN+TP+FN+FP} = \frac{60+63}{160} = 0.77$
- *True positive rate* = $\frac{TP}{TP+FN} = \frac{60}{60+13} = 0.82$ (or *Recall*)
- *False positive rate* = $\frac{FP}{TN+FP} = \frac{24}{24+63} = 0.27$ (or *Specificity*)
- *Precision* = $\frac{TP}{TP+FP} = \frac{60}{60+24} = 0.71$

» Movie Review Example

Using a baseline classifier that just predicts the most common class $+1$, regardless of the input *x*:

	predicted positive	predicted negative
true positive	73	0
true negative	87	0

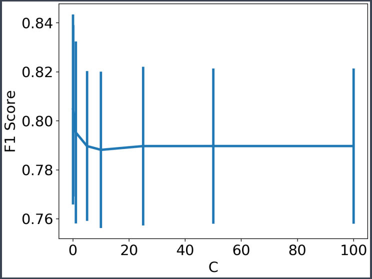
* *Accuracy* = 0.46
* *True positive rate* = $\frac{TP}{TP+FN} = \frac{73}{73+0} = 1.0$
* *False positive rate* = $\frac{FP}{TN+FP} = \frac{20}{20+69} = 0.22$
* *Precision* = $\frac{TP}{TP+FP} = \frac{73}{73+87} = 0.46$

» Movie Review Example

We'd like a single value to plot vs hyperparameters when tuning model. One choice:

- F_1 *score* = $\frac{2TP}{2TP+FN+FP}$ (measures effects of both false positives and false negatives).

but not the only choice. Plotting F_1 score vs penalty parameter C using 5-fold cross-validation:



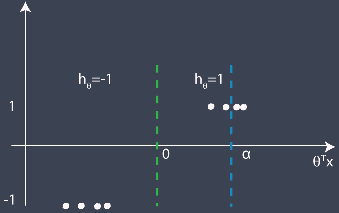
- F_1 score is much the same for all $C \geq 10$, perhaps minimised by $C = 10$

» Movie Review Example

```
from sklearn.svm import LinearSVC
model = LinearSVC(C=1.0).fit(Xtrain, ytrain)
preds = model.predict(Xtest)

from sklearn.metrics import confusion_matrix
print(confusion_matrix(ytest, preds))
from sklearn.metrics import classification_report
print(classification_report(ytest, preds))

from sklearn.dummy import DummyClassifier
dummy = DummyClassifier(strategy="most_frequent").fit(Xtrain, ytrain)
ydummy = dummy.predict(Xtest)
print(confusion_matrix(ytest, ydummy))
print(classification_report(ytest, ydummy))

mean_error=[]
std_error=[]
Ci_range = [0.01, 0.1, 1, 5, 10, 25, 50, 100]
for Ci in Ci_range:
        from sklearn.svm import LinearSVC
        model = LinearSVC(C=Ci)
        from sklearn.model_selection import cross_val_score
        scores = cross_val_score(model, X, y, cv=5, scoring='f1')
        mean_error.append(np.array(scores).mean())
        std_error.append(np.array(scores).std())

import matplotlib.pyplot as plt
plt.rc('font', size=18); plt.rcParams['figure.constrained_layout.use'] = True
plt.errorbar(Ci_range,mean_error,yerr=std_error,linewidth=3)
plt.xlabel('Ci'); plt.ylabel('F1 Score')
plt.show()
```

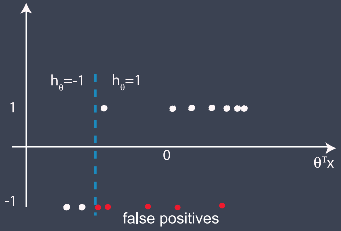
» Decision Threshold

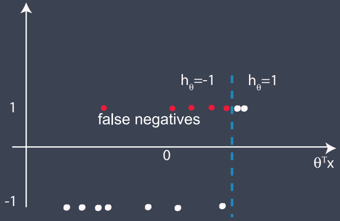
- Linear model: $\hat{y} = sign(\theta^T x)$
- i.e. $\hat{y} = +1$ when $\theta^T x > 0$ and $\hat{y} = -1$ when $\theta^T x < 0$.
- But we can choose a threshold other then 0 i.e. introduce parameter α and predict $\hat{y} = +1$ when $\theta^T x > \alpha$ and $\hat{y} = -1$ when $\theta^T x < \alpha$



- Decision boundary (dashed line) moves to right as α is increased, and to left when α decreased.
- $\alpha = -\infty$ always predict $+1$, $\alpha = +\infty$ always predict -1

» Trade-off Between True Positives and False Positives





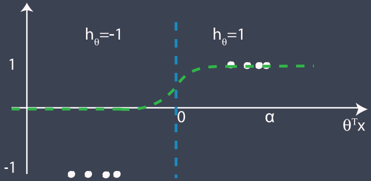
- By varying decision threshold α we can change the balance between false positives and false negatives.
- Sometimes we care more about false negatives than false positives, and vice-versa, and so this gives us freedom to tune classifier
- E.g. if predicting covid exposure notification then to minimise infection spread we want few false negatives (a false negative is an infected person missed by classifier).

» Decision Probability

- Typically, classifiers output a *confidence* value between 0 and 1. Higher means more confident prediction is correct.
- In logistic regression the standard way to map from $\theta^T x$ to a confidence value is the *logistic* or *sigmoid* function:

$$\frac{1}{1+e^{-\theta^T x}} = \frac{e^{\theta^T x}}{e^{\theta^T x}+1}$$

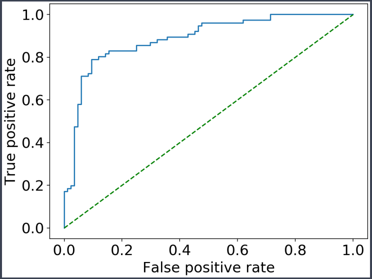
this maps $\theta^T x$ to a value between 0 and 1. Green curve:



- $\theta^T x > \alpha$ when $\frac{1}{1+e^{-\theta^T x}} > \beta = \frac{1}{1+e^{-\alpha}}$. When $\alpha = 0$ then $\beta = 0.5$. Adjusting threshold α is the same as changing the confidence value β above which predict $+1$.

» ROC Curve

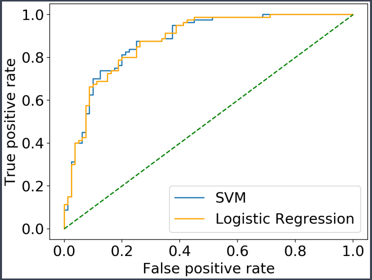
As vary threshold β (or α) the balance between true and false positives varies. A ROC curve is a plot of true positive rate vs false positive rate.
E.g. Movie review example:



- Ideal classifier (100% true positives, 0% false positives) gives point in the top-left corner of ROC plot
- Random classifier is a point on the 45° line (prediction is $+1$ when $z > \alpha$ where z is chosen uniformly at random between -1 and 1, as vary α get 45° line).
- *So want a classifier with ROC curve that comes as close as possible to top-left corner*

» ROC Curve

Can use ROC curve to compare classifiers
E.g. SVM and logistic regression in movie review example:



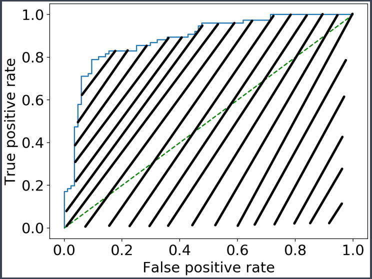
* Both classifiers are much the same, either would be fine.

» ROC Curve

```
import matplotlib.pyplot as plt
plt.rc('font', size=18); plt.rcParams['figure.constrained_layout.use'] = True

from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, test_size=0.2)

from sklearn.svm import LinearSVC
model = LinearSVC(C=1.0).fit(Xtrain, ytrain)

from sklearn.metrics import roc_curve
fpr, tpr, _ = roc_curve(ytest,model.decision_function(Xtest))
plt.plot(fpr,tpr)
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.plot([0, 1], [0, 1], color='green',linestyle='--')
plt.show()
```

» AUC: Area Under Curve

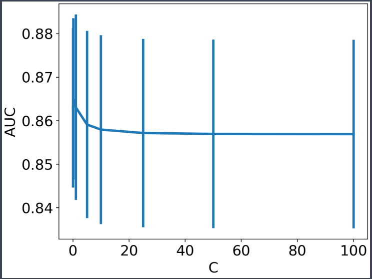
For model tuning we'd like a single value, rather than a curve - so we can plot metric vs hyperparameter value e.g. C



- AUC = *area* under ROC curve
- Ideal classifier: $AUC = 1$. Random classifier: $AUC = 0.5$
- So $0.5 \leq AUC \leq 1$ with closer to 1 better.

» AUC: Area Under Curve

E.g. Movie review example with SVM. Plotting AUC vs penalty weight C using cross-validation:



- AUC is much the same for all $C \geq 10$
- In sklearn change:
  cross_val_score(model, X, y, cv=5, scoring='f1')
  to:
  cross_val_score(model, X, y, cv=5, scoring='auc')

» Model Assessment

Two separate goals:

* *Model selection*: estimating performance of different models in order to choose the best one $\rightarrow$ use cross-validation
* *Model assessment*: having chosen final model, estimate its prediction error on new, previously unseen data i..e generalisation error

Model assessmernt:

* When assessing a model we can use multiple different metrics to capture different aspects of final system e.g. time taken to generate prediction, whether accuracy of prediction changes over time, revenue, user satisfaction.
* Using a baseline for comparison (important!)

» Assessing Generalisation Error

Best practice is to divide our data into two parts:

* Training data used for model selection (using cross-validation to further split this data into training/test data - sometimes this test data is also called *validation* data)
* Test data used to assess prediction accuracy of final model. This is unseen data, never used when training model $\rightarrow$ as soon as we use data to tune the model, the prediction error for that data can be expected to fall and so underestimate the true prediction error for unseen data.
* Dividing data this way is also referred to as *train-validate-test*.

E.g. In competitions training data is released but final evaluation is done by submitting model to a server to be tested against separate data, with multiple resubmissions discouraged.

» Assessing Generalisation Error

And from the horses mouth ...

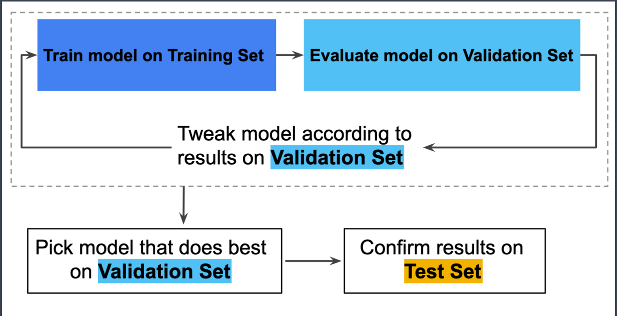

Train model on Training Set
Evaluate model on Validation Set
Tweak model according to results on Validation Set
Pick model that does best on Validation Set
Confirm results on Test Set

Figure 3. A better workflow.

developers.google.com/machine-learning/crash-course/validation/another-partition

- If short of data then separate validation and test data sets might not be possible, but understand that cross-validation may significantly underestimate prediction error for unseen data, so be wary!

» Assessing Generalisation Error

We can use resampling of thetest data to extract a bit more information about the likely generalisation performance.

* Bootstrapping ...