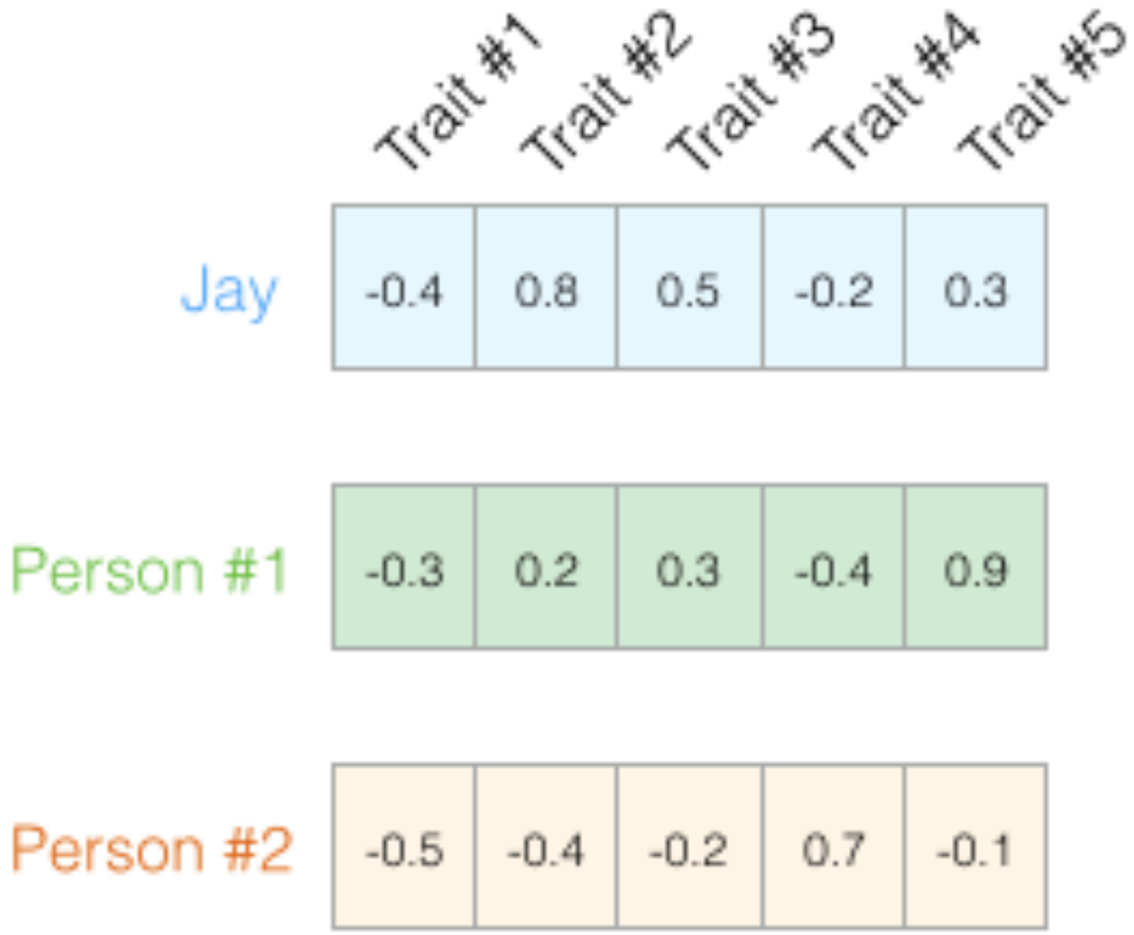
Odds and Ends

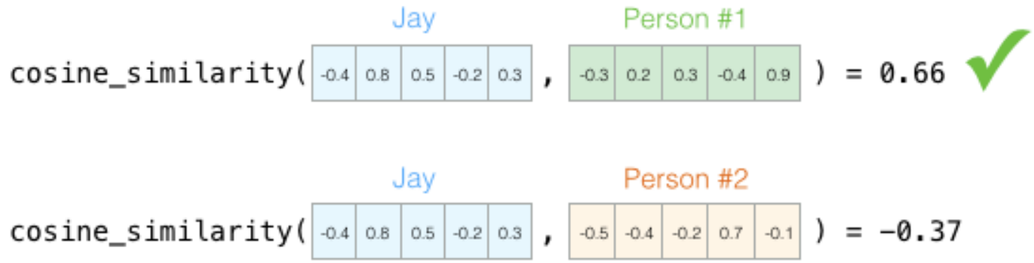
Things to Cover

- Embeddings (for the pros)
- Batch Normalization
- RNN Architectures
- Encoder-Decoder Architectures
- Skip Connections and Resnet
- UNet and Segmentation
- Lab: Pipelines and Embedding

The Basic Idea



- Observe a bunch of people
- **Infer** Personality traits from them
- Vector of traits is called an **Embedding**
- Who is more similar? Jay and who?
- Use Cosine Similarity of the vectors



Images by Jay Alammar

Categorical Data

Example:

Rossmann Kaggle Competition. Rossmann is a 3000 store European Drug Store Chain. The idea is to predict sales 6 weeks in advance.

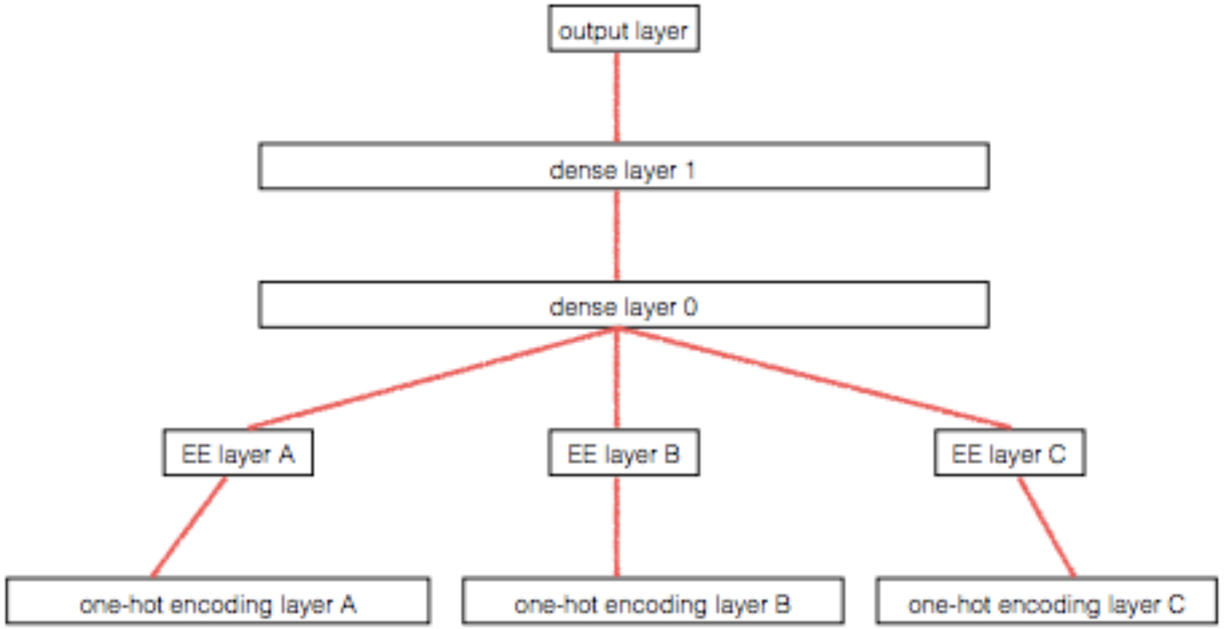
Consider `store_id` as an example. This is a **categorical** predictor, i.e. values come from a finite set.

We usually **one-hot encode** this: a single store is a length 3000 bit-vector with one bit flipped on.

What is the problem with this?

- The 3000 stores have commonalities, but the one-hot encoding does not represent this
- Indeed the dot-product (cosine similarity) of any-2 1-hot bitmaps must be 0
- Would be useful to learn a lower-dimensional **embedding** for the purpose of sales prediction.
- These store "personalities" could then be used in other models (different from the model used to learn the embedding) for sales prediction
- The embedding can be also used for other **tasks**, such as employee turnover prediction

Training an Embedding



- Normally you would do a linear or MLP regression with sales as the target, and both continuous and categorical features
- The game is to replace the 1-hot encoded categorical features by "lower-width" embedding features, for *each* categorical predictor
- This is equivalent to considering a neural network with the output of an additional **Embedding Layer** concatenated in
- The Embedding layer is simply a linear regression

Image from Guo and Berkhahn

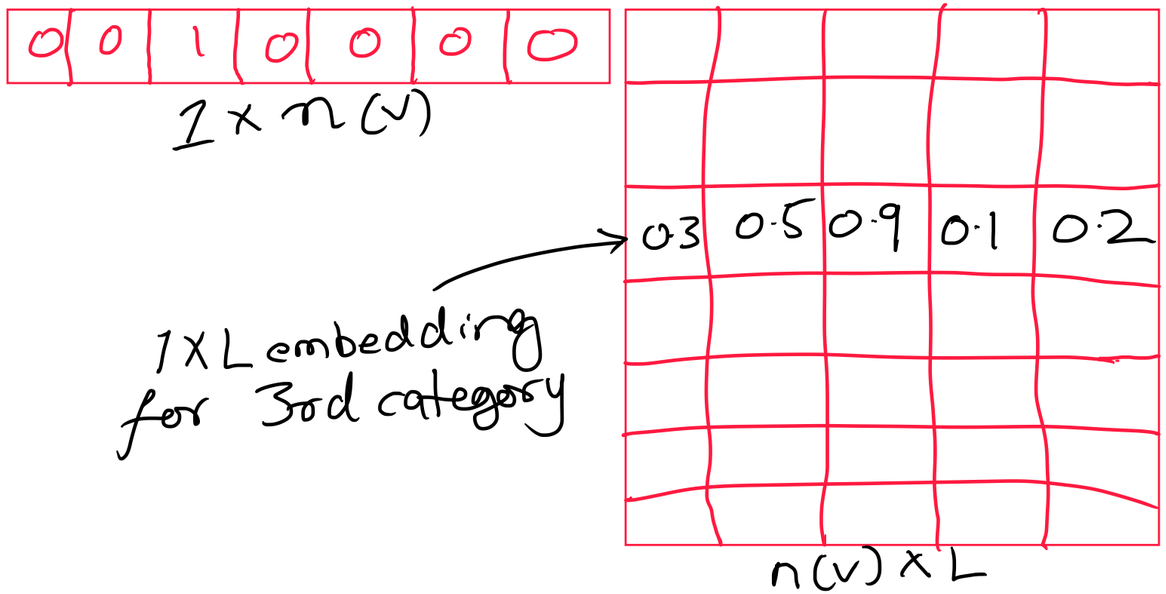
Training an embedding (contd)

A 1-hot vector for a categorical variable v with cardinality $N(v)$ can be written using the Kronecker Delta symbol as

$$v_k = \delta_{jk}, j \in \{1..N(v)\}$$

Then an embedding of width (dimension) L is just a $N(v) \times L$ matrix of weights W_{ij} such that multiplying the kth 1-hot vector by this weight matrix by picks out the kth row of weights (see right)

But how do we find these weights? We fit for them with the rest of the weights in the MLP!



Keras Functional API: simple MLP

```
rom keras.layers import Input, Dense
from keras.models import Model
# This returns a tensor
inputs = Input(shape=(784,))
# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)
# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels)  # starts training
```

```
def build_keras_model():
    input_cat = []
    output_embeddings = []
    for k in cat_vars+nacols_cat: #categoricals plus NA booleans
        input_1d = Input(shape=(1,))
        output_1d = Embedding(input_cardinality[k], embedding_cardinality[k], name='{}_embedding'.format(k))(input_1d)
        output = Reshape(target_shape=(embedding_cardinality[k],))(output_1d)
        input_cat.append(input_1d)
        output_embeddings.append(output)

    main_input = Input(shape=(len(cont_vars),), name='main_input')
    output_model = Concatenate()([main_input, *output_embeddings])
    output_model = Dense(1000, kernel_initializer="uniform")(output_model)
    output_model = Activation('relu')(output_model)
    output_model = Dense(500, kernel_initializer="uniform")(output_model)
    output_model = Activation('relu')(output_model)
    output_model = Dense(1)(output_model)

    kmodel = KerasModel(
        inputs=[*input_cat, main_input],
        outputs=output_model
    )
    kmodel.compile(loss='mean_squared_error', optimizer='adam')
    return kmodel

def fitmodel(kmodel, Xtr, ytr, Xval, yval, epochs, bs):
    h = kmodel.fit(Xtr, ytr, validation_data=(Xval, yval),
                       epochs=epochs, batch_size=bs)
    return h
```

Embedding is just a linear regression

So why are we giving it another name?

- it is usually to a lower dimensional space
- traditionally we have done linear dimensional reduction through PCA or SVD and truncation, but sparsity can throw a spanner into the works
- we train the weights of the embedding regression using SGD, along with the weights of the downstream task (here fitting the rating)
- the embedding can be used for alternate tasks, such as finding the similarity of users.

See how Spotify does all this..

Usages

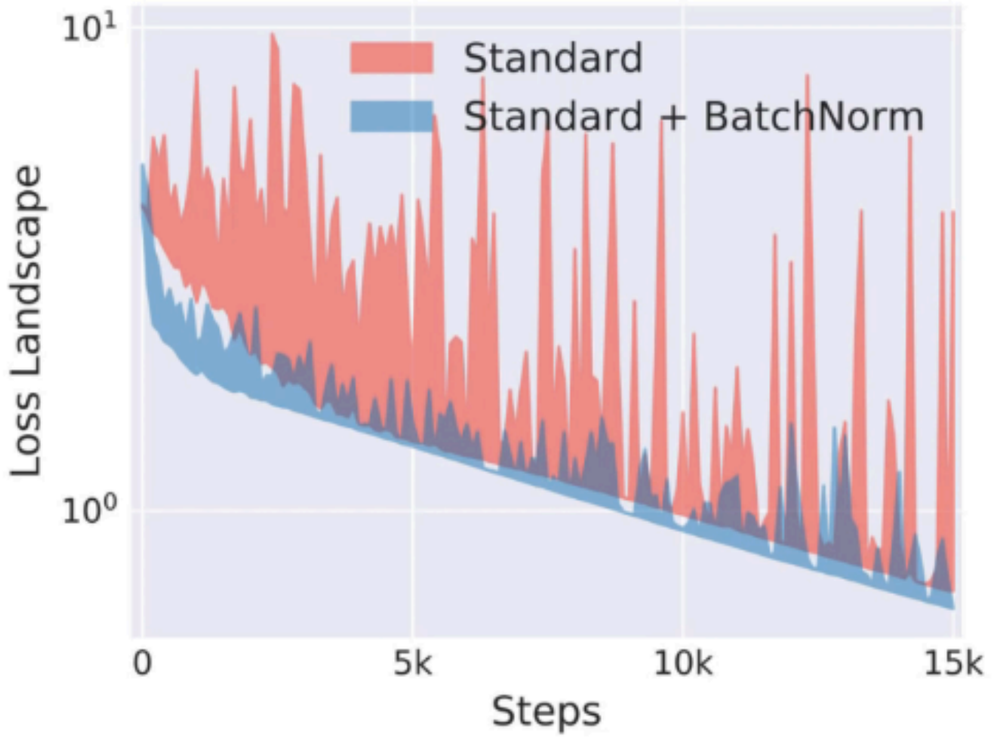
- the pre-trained word2vec and other embeddings (such as GloVe) are used everywhere in NLP today
- the ideas have been used elsewhere as well. AirBnB and Anghami model sequences of listings and songs using word2vec like techniques
- Alibaba and Facebook use word2vec and graph embeddings for recommendations and social network analysis.
- But word2vec cannot disambiguare the context of "bank" in "*open a bank account*" from that in "*on the river bank*"
- **ELMo** looks at the entire sentence before assigning a word an embedding. It is a Bi-Directional LSTM Model trained on language modeling and customized by training on specific tasks
- **BERT** trains a transformer based model on language modeling via masking future words and via predicting futire and past sentences. This provides SOTA results on many downstream tasks by either fine-tuning it or using pre-trained embeddings.

Batch Normalization

"Normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1."

- can be done before the activation(relu) or after
- current practice is to do it after
- the scale of the transformation and the mean are fit for on a mini-batch!

- allows one to train with a higher learning rate. It ensures that the activations are hit in a region where gradient is large.
- is done on a per batch basis, usually in the channel (-1)direction (similar to spatial dropout)
- be careful if you are retraining layers of a batchnormalized base model in transfer learning. If your dataset is from a different distribution, you want the training take this into account. (see https://github.com/keras-team/keras/pull/9965)
- works by allowing other neural weights to focus on training the layer, rather than on scaling



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

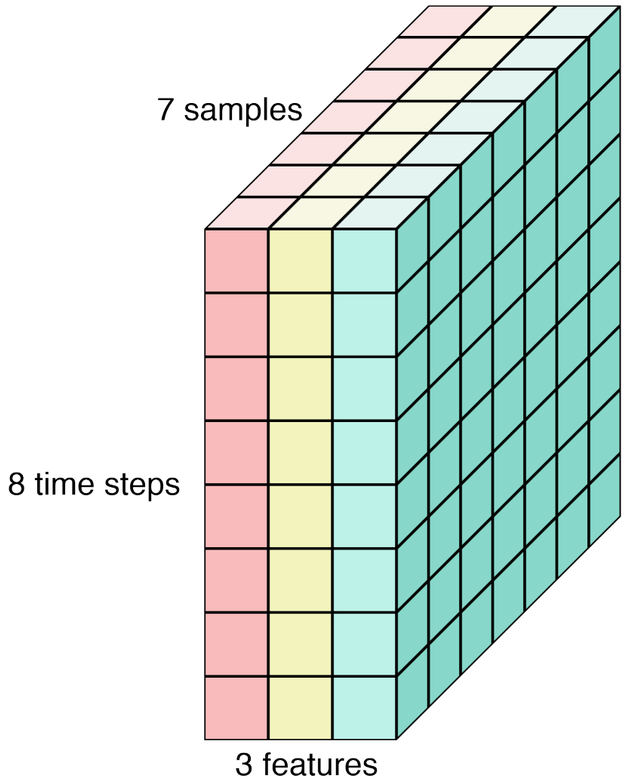
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

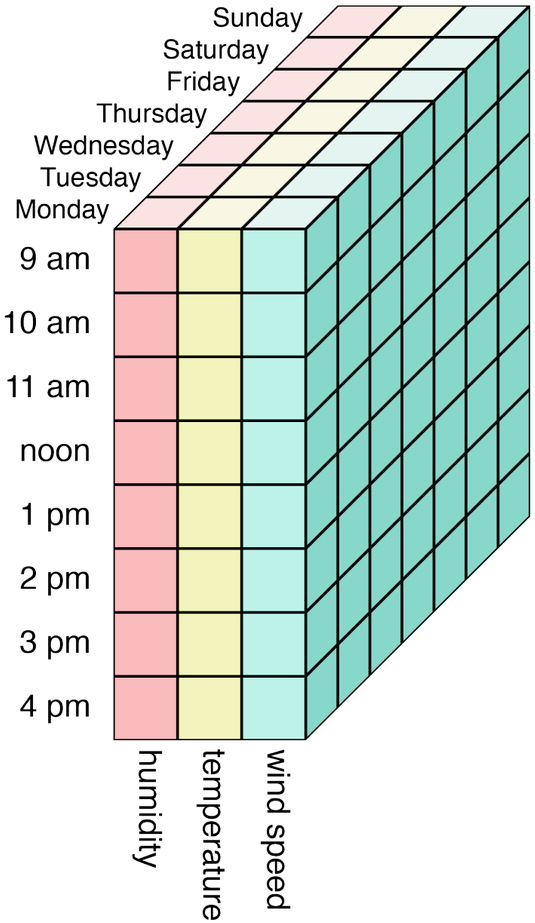
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

BN trains much more smoothly, allowing us to use a higher learning rate.

RNN Architectures

Data layout: *numsamples x timesteps x features*





Sentiment analysis

Many->1 LSTM with each sample a review, time*steps the max*len of the reviews, and features the embedding size. One output.

```
X_train = tokenizer.texts_to_matrix(X_train)
X_train = sequence.pad_sequences(X_train, maxlen=config.maxlen)
model = Sequential()
model.add(Embedding(config.vocab_size,
                    config.embedding_dims,
                    input_length=config.maxlen))
model.add(LSTM(config.hidden_dims, activation="sigmoid"))
model.add(Dense(1, activation='sigmoid'))
```

Character generation: *sentlen x maxlen x vocablen*

```
#predict one char from the previous n chars
for i in range(0, len(text) - config.maxlen, config.step):
    sentences.append(text[i: i + config.maxlen])
    next_chars.append(text[i + config.maxlen])
x = np.zeros((len(sentences), config.maxlen, vocablen))
y = np.zeros((len(sentences), vocablen))
for i, sentence in enumerate(sentences):
    for t, char in enumerate(sentence):
        x[i, t, char_indices[char]] = 1
    y[i, char_indices[next_chars[i]]] = 1
model = Sequential()
model.add(GRU(config.hidden_nodes, input_shape=(config.maxlen, vocablen)))
model.add(Dense(len(chars), activation='softmax'))
```

many -> 1 but constructed from 1 document

Captions: many -> many, *numsamples x maxlen x embedlen*

```
dense_input = BatchNormalization(axis=-1)(image_model.output)
image_dense = Dense(units=embedding_size)(dense_input)  # FC layer
# Add a timestep dimension to match LSTM's input size
image_embedding = RepeatVector(1)(image_dense)
image_input = image_model.input
# separate sentence unput
sentence_input = Input(shape=[None])
word_embedding = Embedding(input_dim=vocab_size, output_dim=embedding_size)(sentence_input)
sequence_input = Concatenate(axis=1)([image_embedding, word_embedding])
input_ = BatchNormalization(axis=-1)(sequence_input)
lstm_out = LSTM(units=lstm_output_size, return_sequences=True,
            dropout=dropout_rate, recurrent_dropout=dropout_rate)(input_)
sequence_output = TimeDistributed(Dense(units=vocab_size))(lstm_out)
model = Model(inputs=[image_input, sentence_input], outputs=sequence_output)
```

Skip Connections

- the architectures of LSTMs and GRUs are created to keep memory, to prevent repeated use of non-linearities driving the gradient to 0
- this is a general problem in any sufficiently deep network
- Haiming Ke came up wuth a great solution for this: Residual Networks , or Resnet
- Fitting on the residuals is like boosting.

The idea is simply to add the values in a layer with a downstream layer.

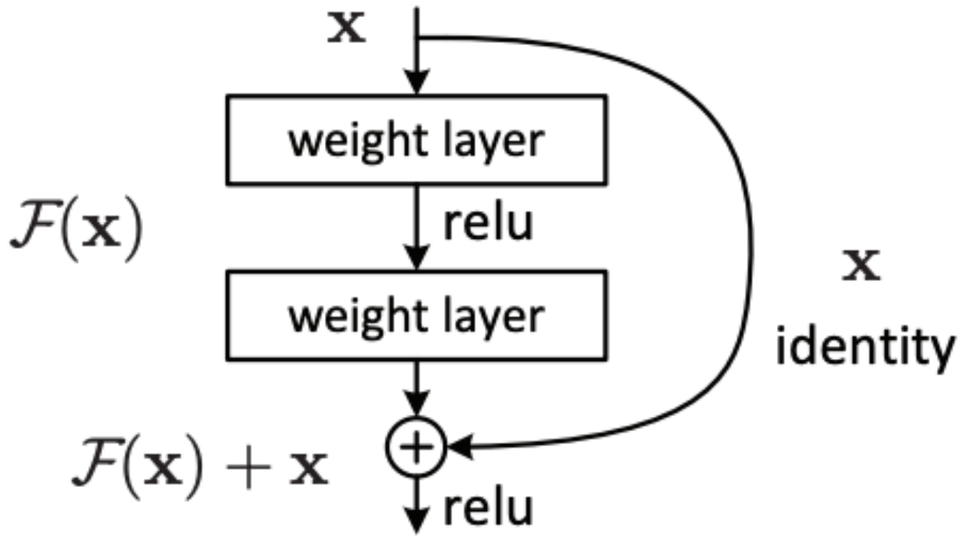
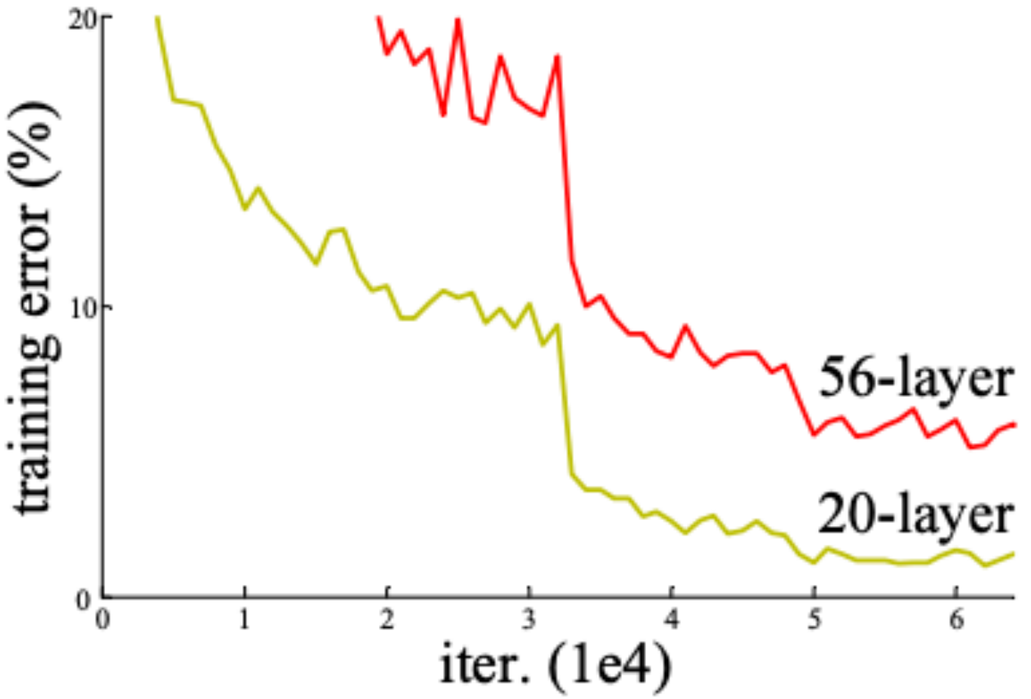


Figure 2. Residual learning: a building block.



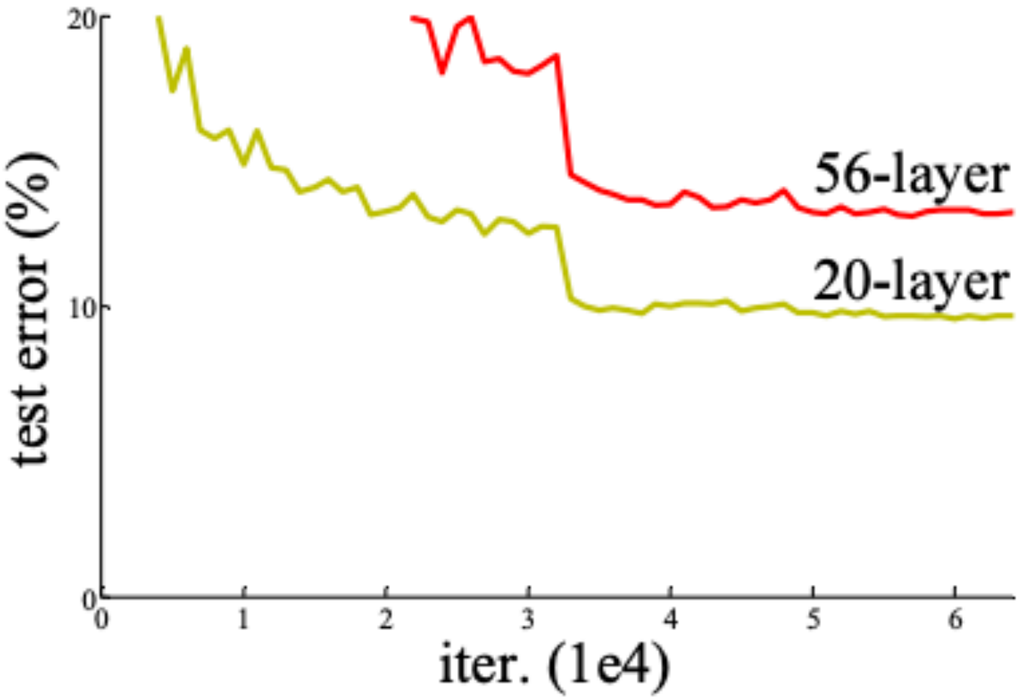
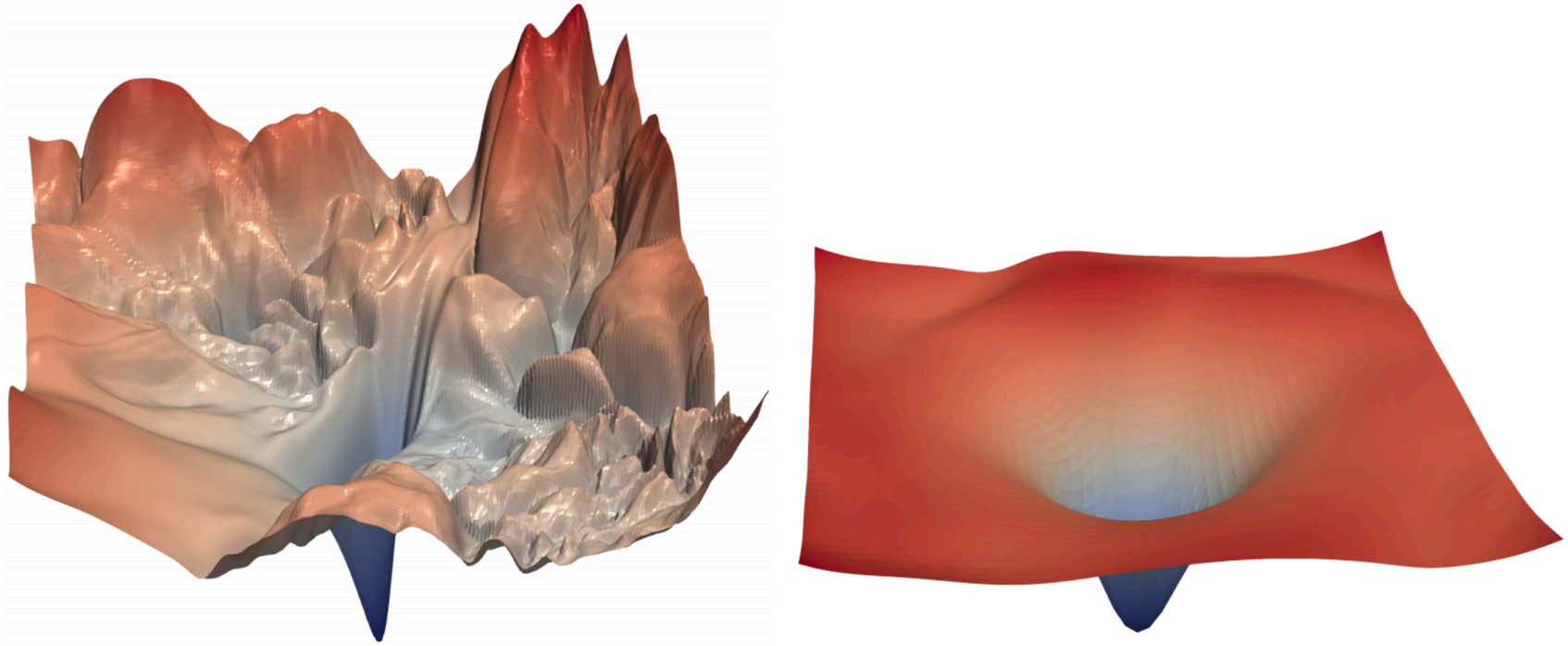


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Why does it work? Makes loss landscape smoother

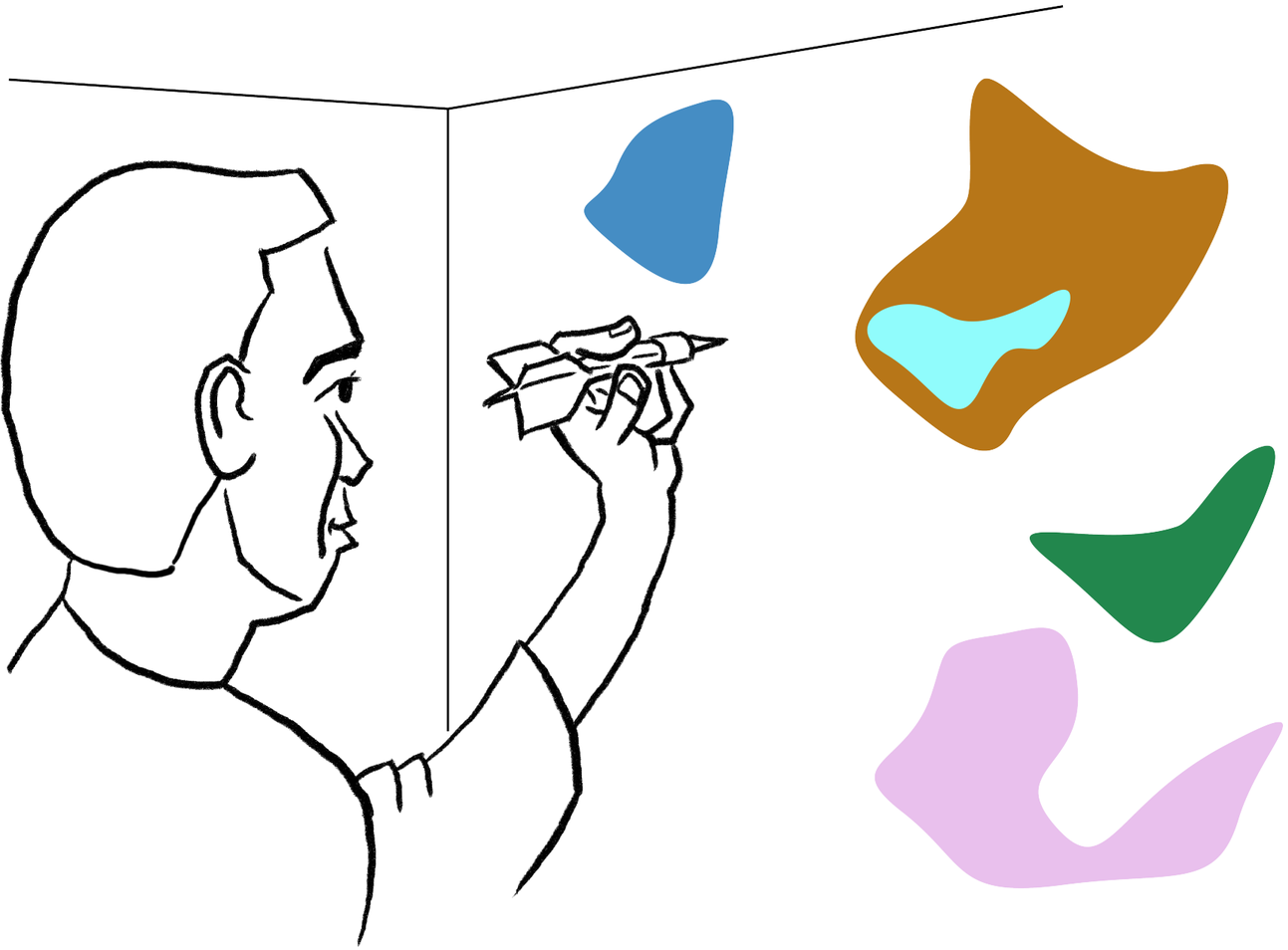


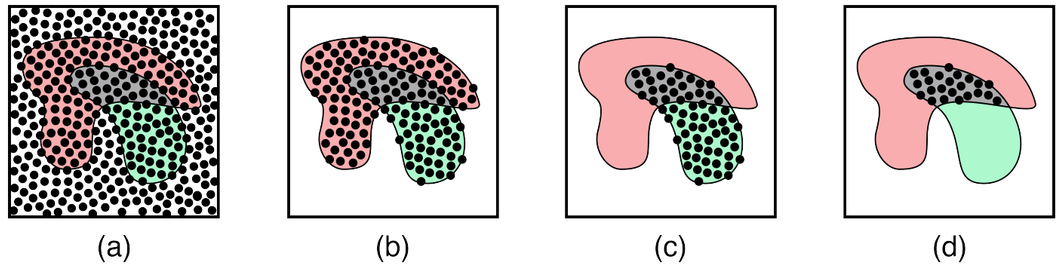
(a) without skip connections

(b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

Throwing darts, uniformly

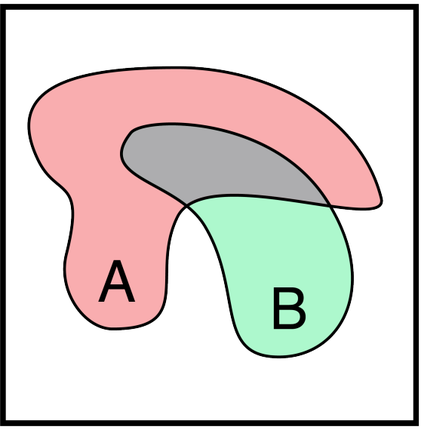




Throwing darts at the wall to find P(A|B). (a) Darts striking the wall. (b) All the darts in either A or B. (c) The darts only in B. (d) The darts that are in the overlap of A and B.

(pics like these from Andrew Glassner's book)

Conditional Probability



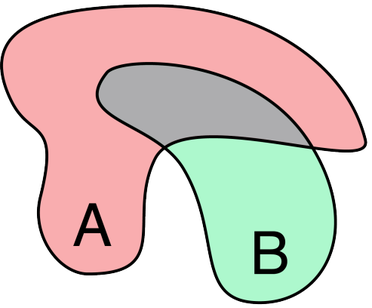
$$P(A|B) = \frac{\text{shape}}{\text{shape}}$$

conditional probability tells us the chance that one thing will happen, given that another thing has already happened. In this case, we want to know the probability that our dart landed in blob A, given that we already know it landed in blob B.

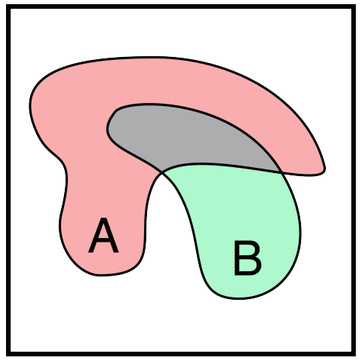
Other conditional and joint

Left: the other conditional

Below: the joint probability $p(A, B)$, the chance that any randomly-thrown dart will land in both A and B at the same time.



$$P(B|A) = \frac{\text{}}{\text{}}$$

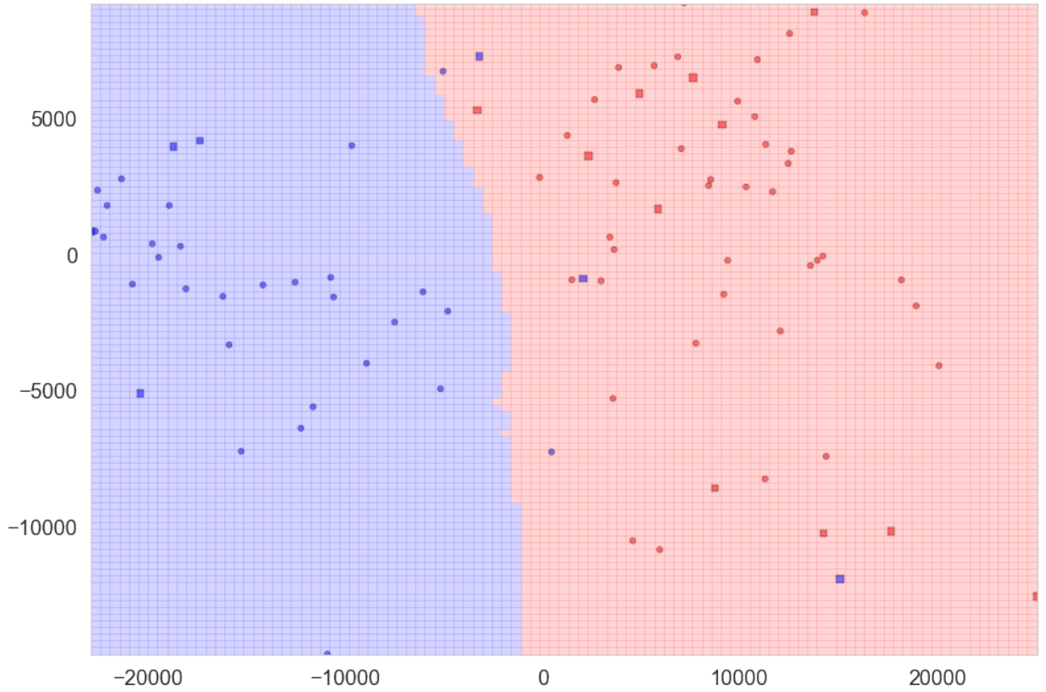


$$P(A,B) = \frac{\text{}}{\square}$$

EVALUATING AND COMPARING CLASSIFIERS

Predicted

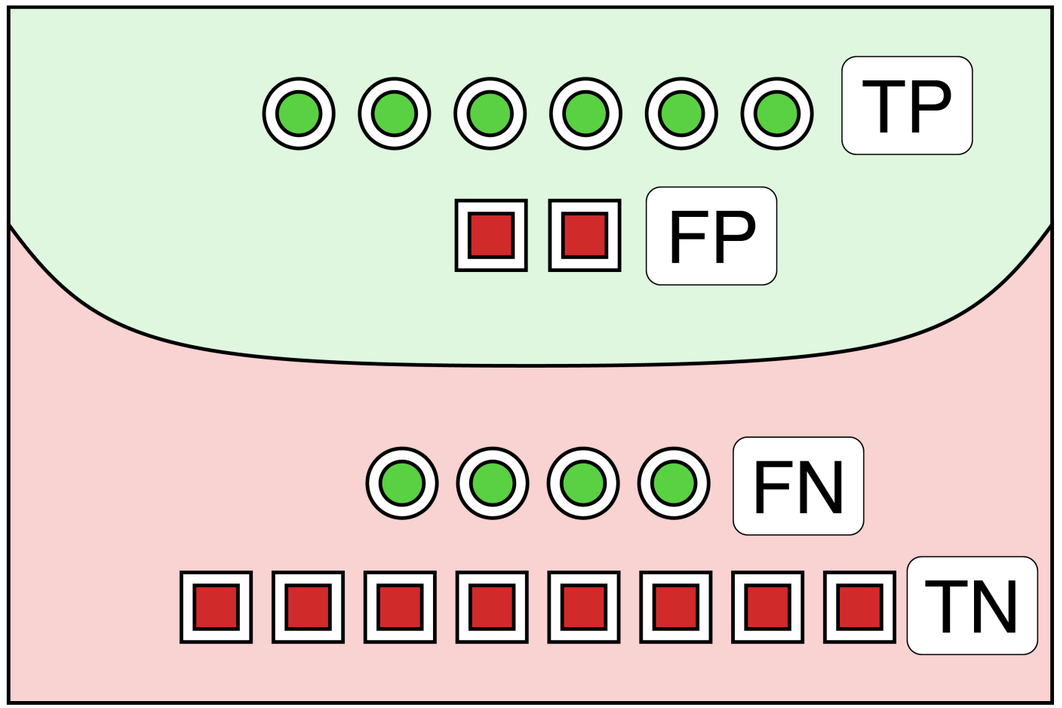
Observed	0	1	
0	TN True Negative	FP False Positive	ON Observed Negative
1	FN False Negative	TP True Positive	OP Observed Positive
	PN Predicted Negative	PP Predicted Positive	



Metrics (from Glassner)

- accuracy is a number from 0 to 1. It's a general measure of how often the prediction is correct.
- Precision (also called positive predictive value, or PPV) tells us the percentage of our samples that were properly labeled "positive," relative to all the samples we labeled as "positive." Numerically, it's the value of TP relative to TP+FP. In other words, precision tells us how many of the "positive" predictions were really positive.
- recall, (also called sensitivity, hit rate, or true positive rate). This tells us the percentage of the positive samples that we correctly labeled.
- F1 score is the harmonic mean of precision and recall. Generally speaking, the f1 score will be low when either precision or recall is low, and will approach 1 when both measures also approach 1.

Metrics (example)



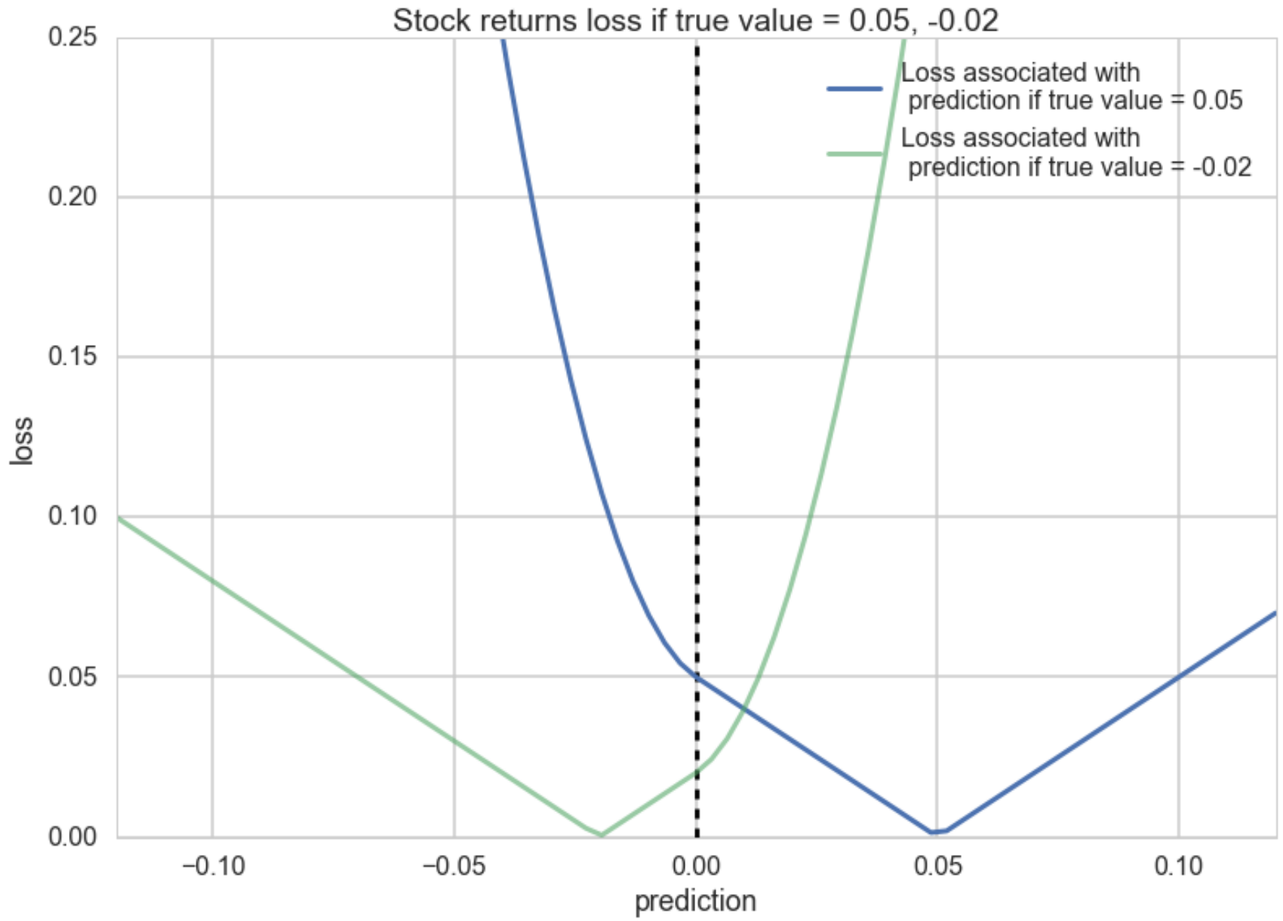
Recall	TPR	$\frac{TP}{TP+FN}$		=	$\frac{6}{6+4}$	= 6/10	= 0.6
Specificity	TNR	$\frac{TN}{TN+FP}$		=	$\frac{8}{8+2}$	= 8/10	= 0.8
Precision	PPV	$\frac{TP}{TP+FP}$		=	$\frac{6}{6+2}$	= 6/8	= 0.75
Negative Predictive Value	NPV	$\frac{TN}{TN+FN}$		=	$\frac{8}{8+4}$	= 8/12	≈ 0.66
False Negative Rate	FNR	$\frac{FN}{FN+TP}$		=	$\frac{4}{4+6}$	= 4/10	= 0.4
False Positive Rate	FPR	$\frac{FP}{FP+TN}$		=	$\frac{2}{2+8}$	= 2/10	= 0.2
False Discovery Rate	FDR	$\frac{FP}{TP+FP}$		=	$\frac{2}{2+6}$	= 2/8	= 0.25
True Discovery Rate	TDR	$\frac{FN}{TN+FN}$		=	$\frac{4}{4+8}$	= 4/12	≈ 0.33
Accuracy	ACC	$\frac{TP+TN}{TP+FP+TN+FN}$		=	$\frac{6+8}{6+2+4+8}$	= 14/20	= 0.7
F1 Score	F1	$\frac{2\,TP}{2\,TP+FP+FN}$		=	$\frac{2*6}{(2*6)+2+4}$	= 12/18	≈ 0.66

Decision Theory Model Comparison

Predictions (or actions based on predictions) are described by a utility or loss function, whose values can be computed given the observed data.

Custom Loss: Stock Market Returns


```
def stock_loss(stock_return, pred, alpha = 100.):
    if stock_return * pred < 0:
        #opposite signs, not good
        return alpha*pred**2 - np.sign(stock_return)*pred \
                        + abs(stock_return)
    else:
        return abs(stock_return - pred)
```



The two risks

There are *two risks in learning* that we must consider, one to *estimate probabilities*, which we call **estimation risk**, and one to *make decisions*, which we call **decision risk**.

The **decision loss** $l(y, a)$ or **utility** $u(l, a)$ (profit, or benefit) in making a decision a when the predicted variable has value y. For example, we must provide all of the losses l(no-cancer, biopsy), l(cancer, biopsy), l(no-cancer, no-biopsy), and l(cancer, no-biopsy). One set of choices for these losses may be 20, 0, 0, 200 respectively.

Classification Risk

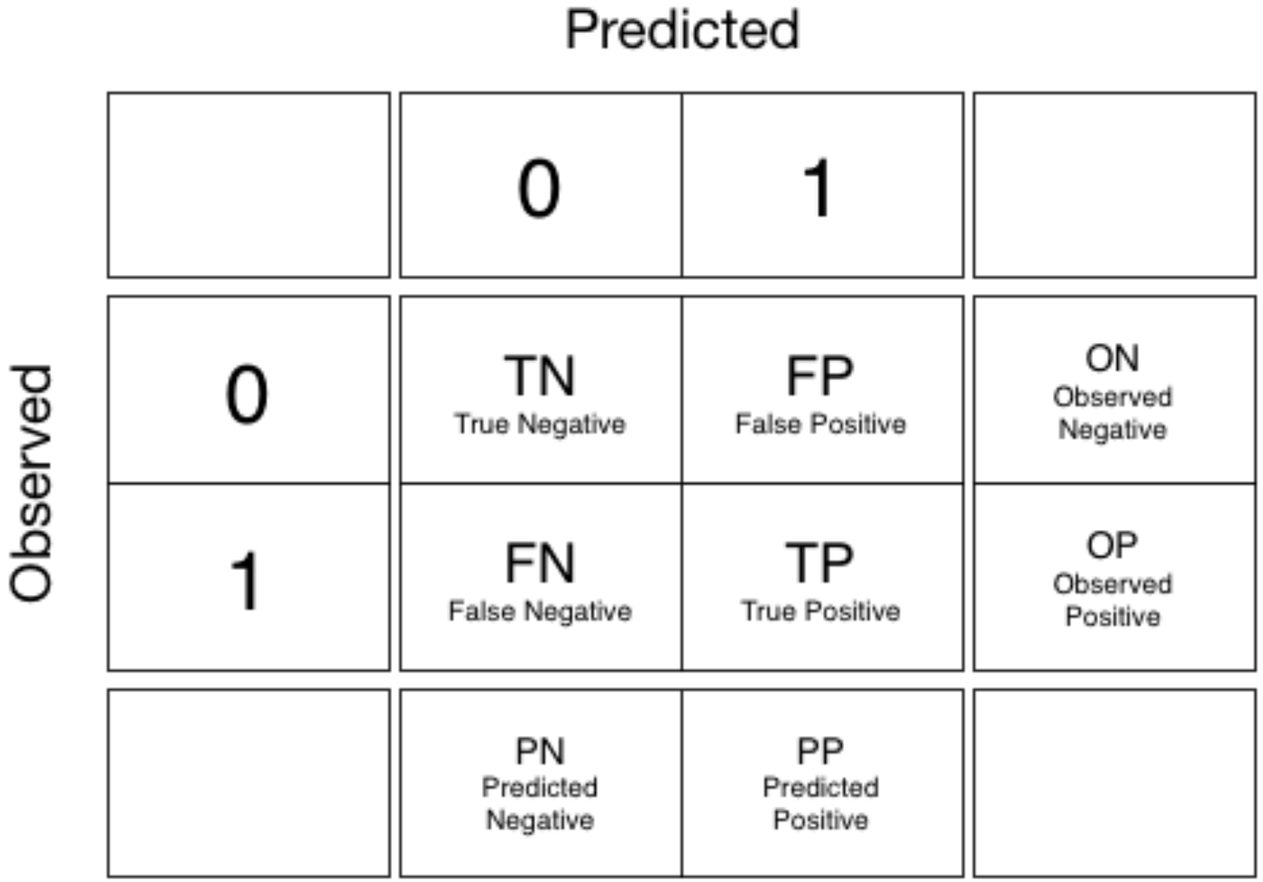
$$R_a(x) = \sum_y l(y, a(x))p(y|x)$$

That is, we calculate the **predictive averaged risk** over all choices y, of making choice a for a given data point.

Overall risk, given all the data points in our set:

$$R(a) = \int dx p(x) R_a(x)$$

Two class Classification



$$R_a(x) = l(1,g)p(1|x) + l(0,g)p(0|x).$$

Then for the "decision" $a = 1$ we have:

$$R_1(x) = l(1,1)p(1|x) + l(0,1)p(0|x),$$

and for the "decision" $a = 0$ we have:

$$R_0(x) = l(1,0)p(1|x) + l(0,0)p(0|x).$$

Now, we'd choose 1 for the data point at x if:

$$R_1(x) < R_0(x).$$

$$P(1|x)(l(1,1) - l(1,0)) < p(0|x)(l(0,0) - l(0,1))$$

So, to choose '1', the Bayes risk can be obtained by setting:

$$p(1|x) > rP(0|x) \implies r = \frac{l(0,1) - l(0,0)}{l(1,0) - l(1,1)}$$

$$P(1|x) > t = \frac{r}{1+r}.$$

One can use the prediction cost matrix corresponding to the consufion matrix

$$r == \frac{c_{FP} - c_{TN}}{c_{FN} - c_{TP}}$$

If you assume that True positives and True negatives have no cost, and the cost of a false positive is equal to that of a false positive, then $r = 1$ and the threshold is the usual intuitive $t = 0.5$.

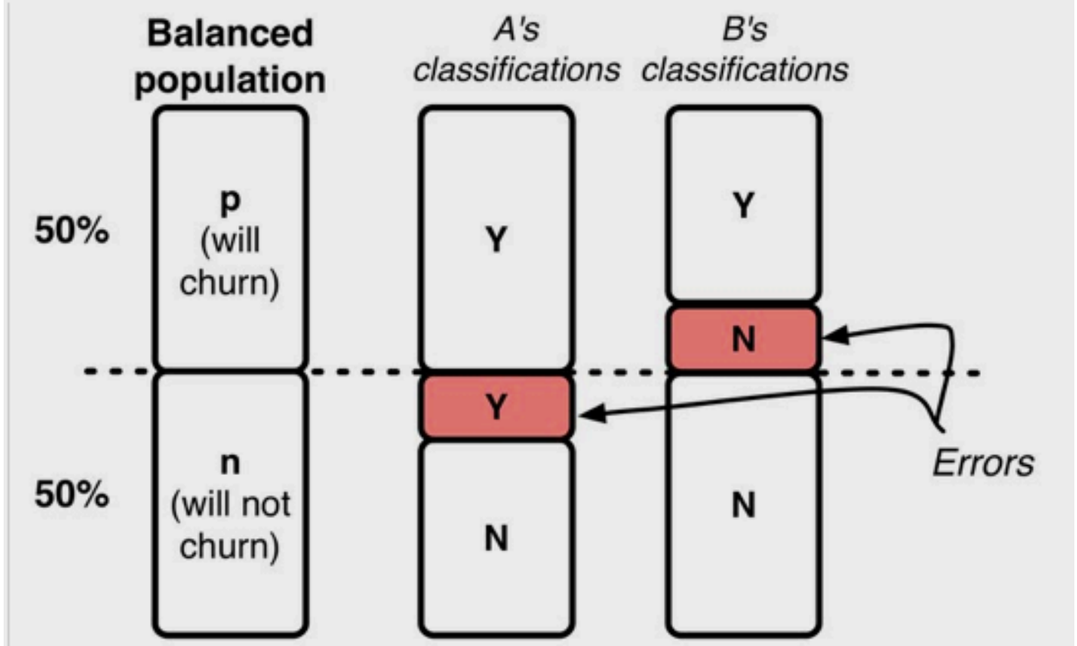
		Predicted	
		0	1
Observed	0	TNC True Negative Cost	FPC False Positive Cost
	1	FNC False Negative Cost	TPC True Positive Cost

COMPARING CLASSIFERS

Telecom customer Churn data set from @YhatHQ[<]

VMail Message	Day Mins	Day Calls	Day Charge	Eve Mins	Eve Calls	Eve Charge	Night Mins	Night Calls	Night Charge	Intl Mins	Intl Calls	Intl Charge	CustServ Calls	Churn?
25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3	2.70	1	False.
26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3	3.70	1	False.
0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5	3.29	0	False.
0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	2	False.
0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	3	False.

[<] http://blog.yhathq.com/posts/predicting-customer-churn-with-sklearn.html

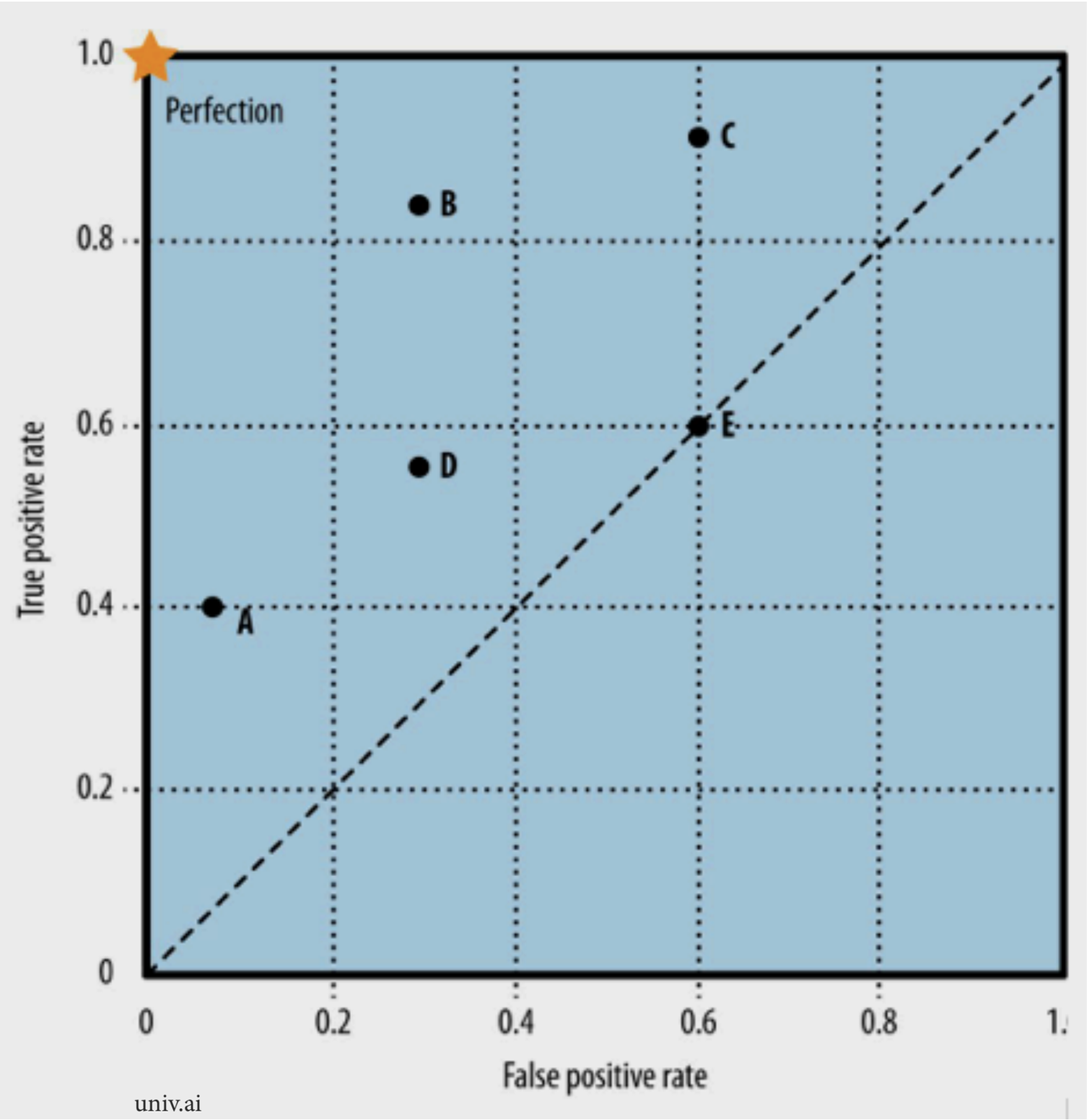


univ.ai

ASYMMETRIC CLASSES

- A has large FP[#]
- B has large FN.
- On asymmetric data sets, A will do very bad.
- Both downsampling and unequal classes are used in practice

[#] figure from Data Science for Business, Foster et. al.



univ.ai

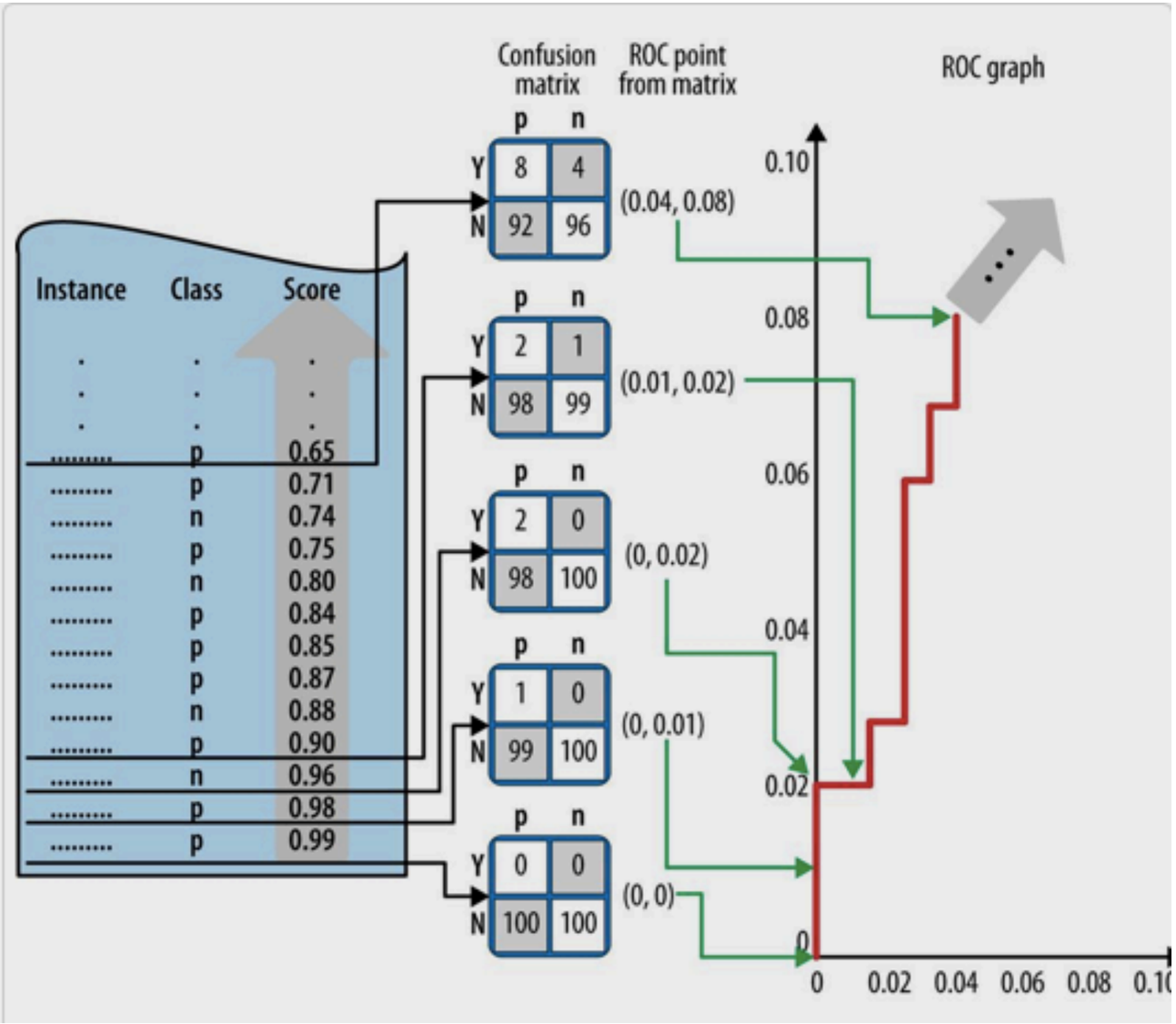
ROC SPACE[+]

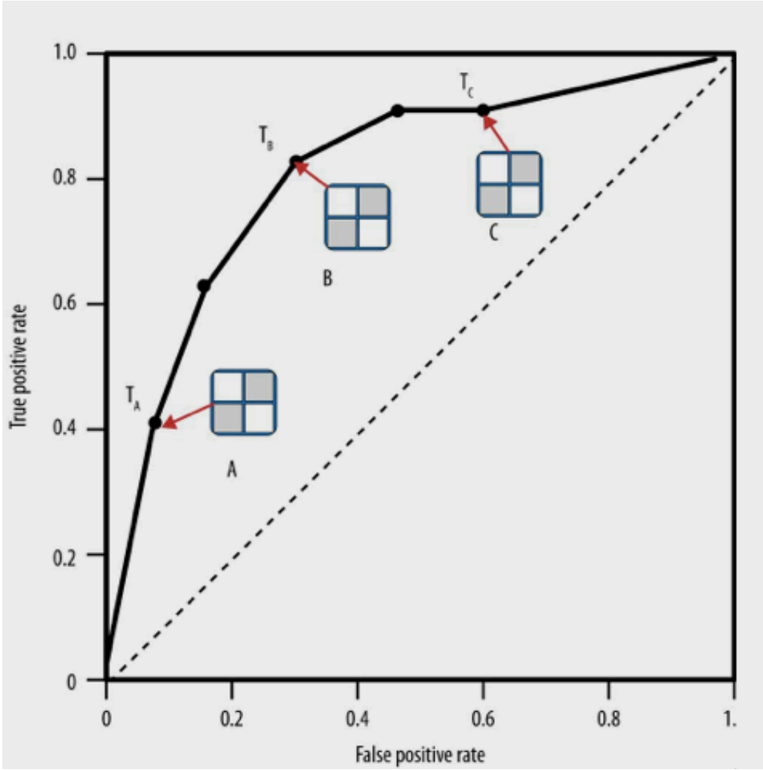
$$TPR = \frac{TP}{OP} = \frac{TP}{TP + FN}.$$

$$FPR = \frac{FP}{ON} = \frac{FP}{FP + TN}$$

		Predicted		
		0	1	
Observed	0	TN True Negative	FP False Positive	ON Observed Negative
	1	FN False Negative	TP True Positive	OP Observed Positive
		PN Predicted Negative	PP Predicted Positive	

[+] this+next fig: Data Science for Business, Foster et. al.

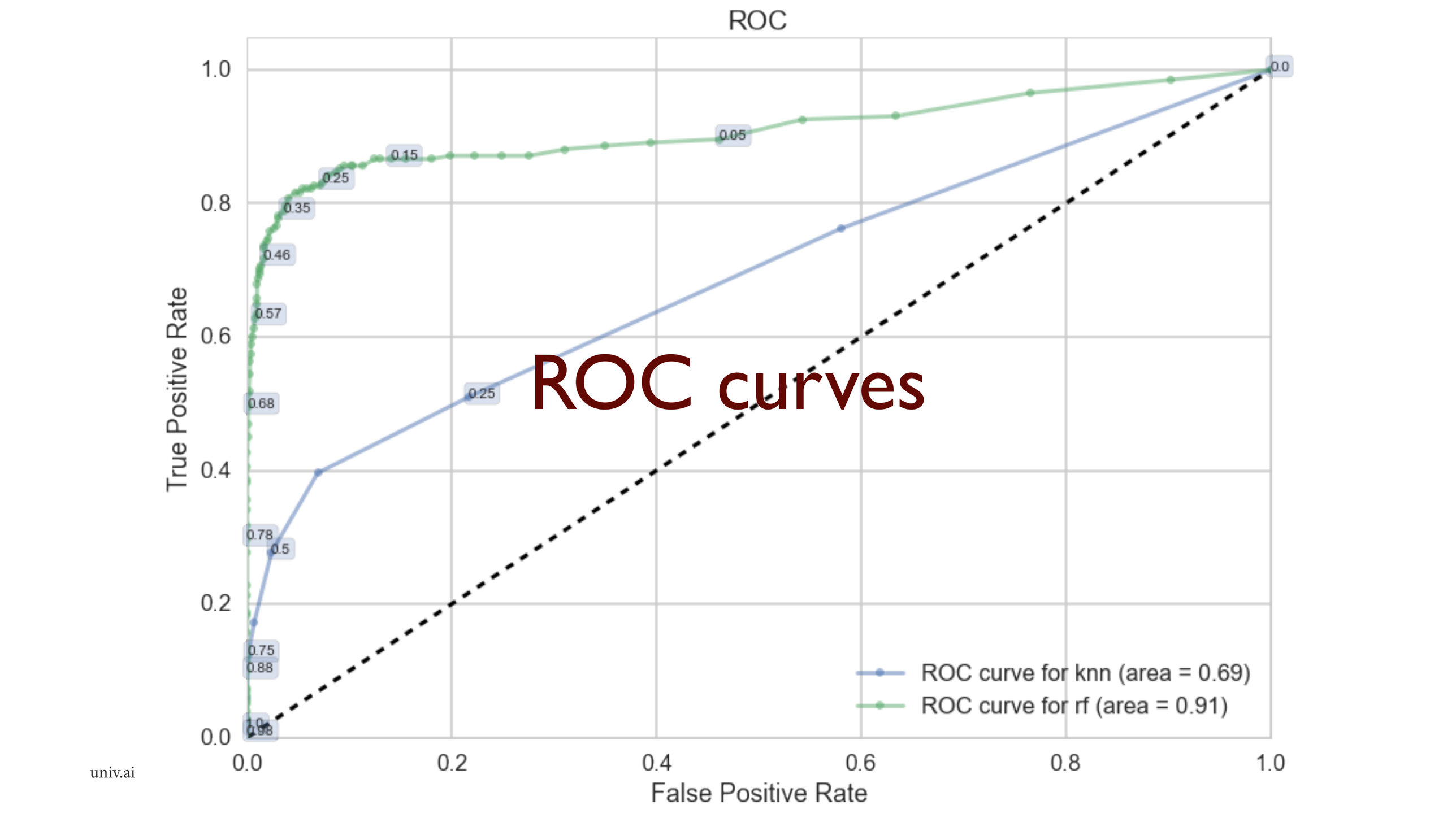

Confusion matrix
ROC point from matrix
ROC graph
Instance
Class
Score
p n
Y 8 4
N 92 96
(0.04, 0.08)
p n
Y 2 1
N 98 99
(0.01, 0.02)
p n
Y 2 0
N 98 100
(0, 0.02)
p n
Y 1 0
N 99 100
(0, 0.01)
p n
Y 0 0
N 100 100
(0, 0)
p 0.65
p 0.71
n 0.74
p 0.75
n 0.80
p 0.84
p 0.85
p 0.87
n 0.88
p 0.90
n 0.96
p 0.98
p 0.99
0.10
0.08
0.06
0.04
0.02
0
0 0.02 0.04 0.06 0.08 0.10


True positive rate
False positive rate
1.0
0.8
0.6
0.4
0.2
0
0 0.2 0.4 0.6 0.8 1.
T_A
T_B
T_C
A
B
C

ROC Curve

ROC CURVE

- Rank test set by prob/score from highest to lowest
- At beginning no +ives
- Keep moving threshold
- confusion matrix at each threshold


ROC
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.05
0.15
0.25
0.35
0.46
0.57
0.68
0.25
0.78
0.5
0.75
0.88
1.0
0.98
ROC curve for knn (area = 0.69)
ROC curve for rf (area = 0.91)

ROC curves

ASYMMETRIC CLASSES

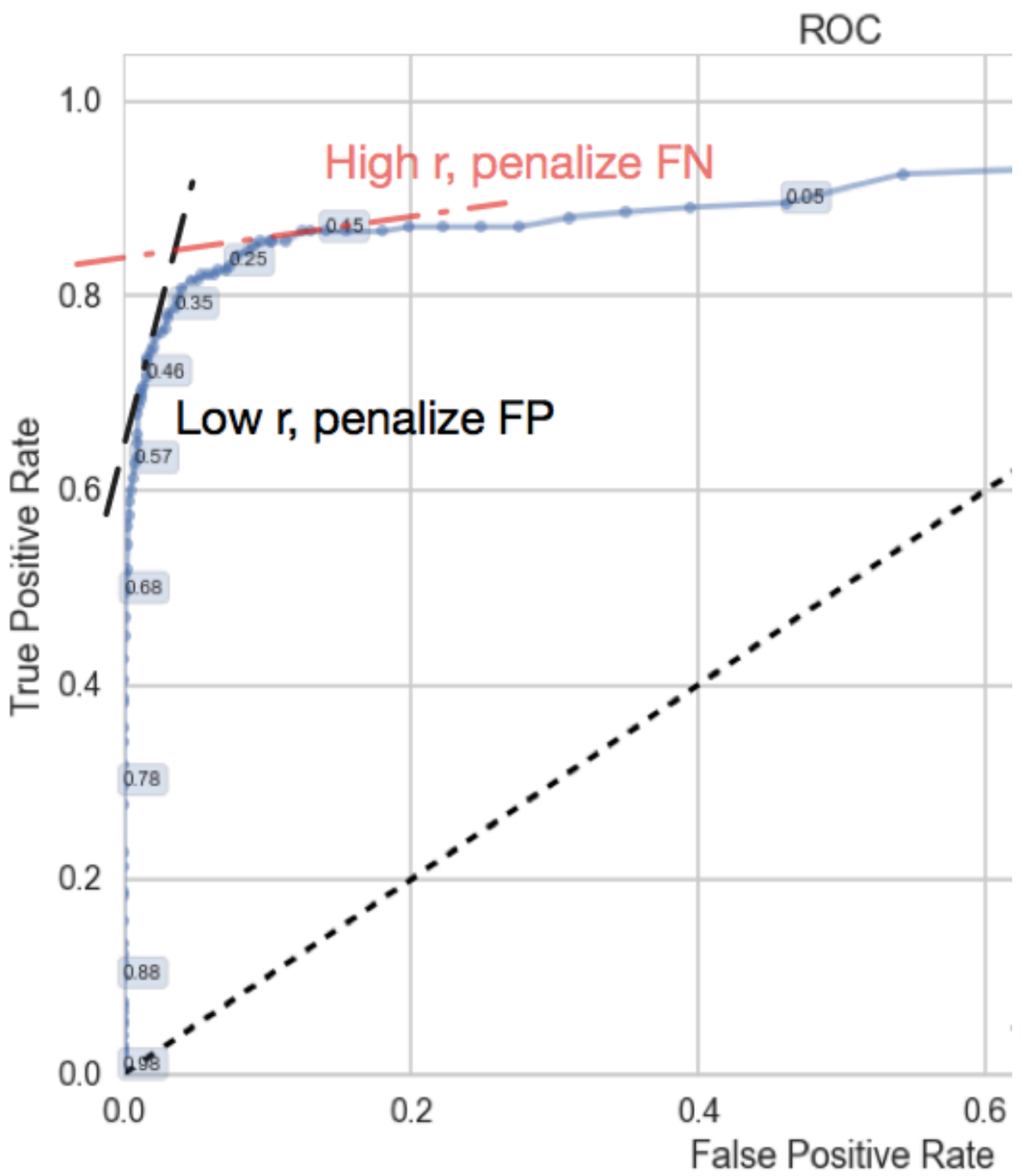
$$r = \frac{l_{FN}}{l_{FP}}$$

We look for lines with slope

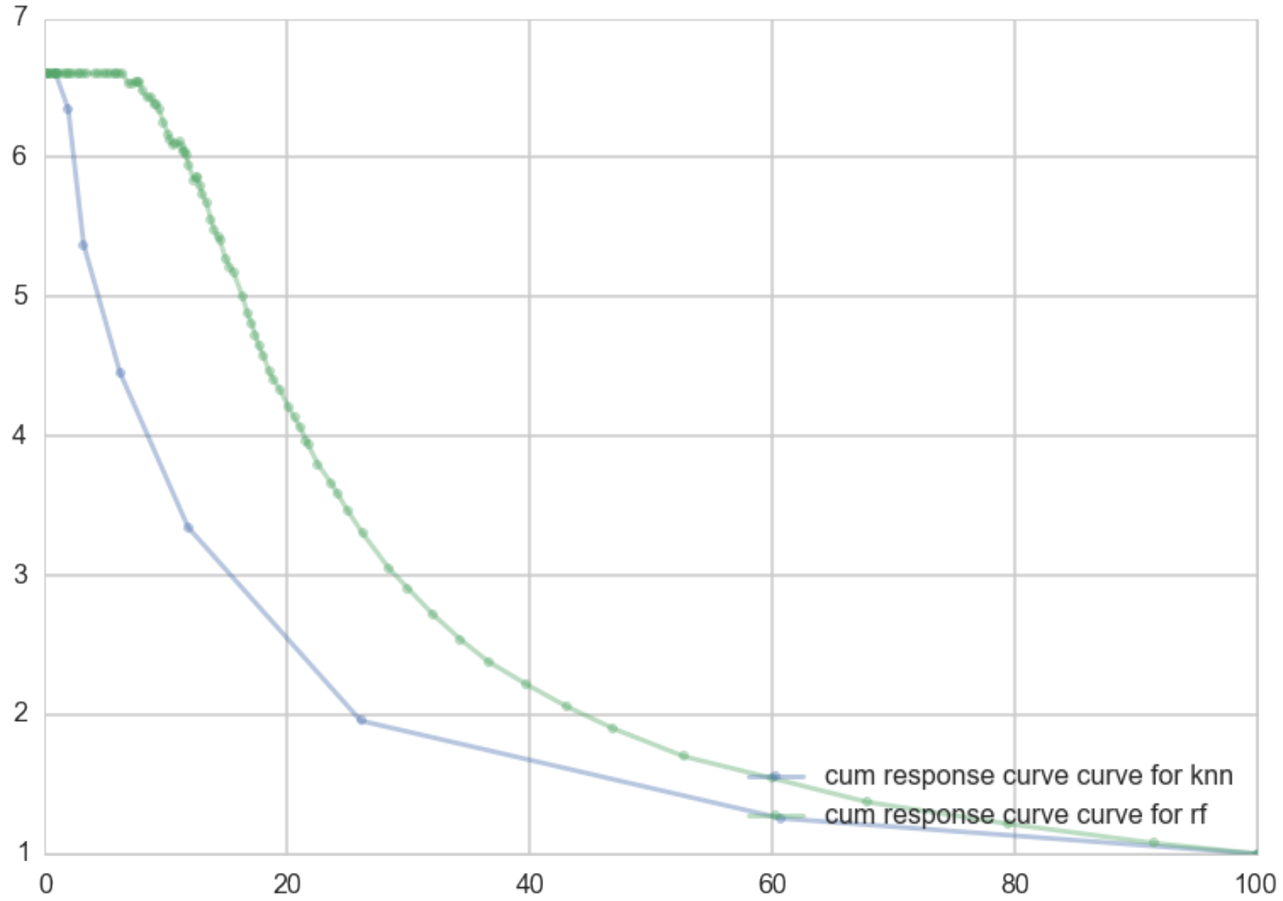
$$\frac{p(0)}{r\,p(1)} = \frac{p(-)}{r\,p(+)} \sim \frac{10}{r}$$

Large r penalizes FN.

Churn and Cancer u dont want FN: an uncaught churner or cancer patient (P=churn/cancer)



Lift Curve



- Rank test set by prob/score from highest to lowest
- Calculate TPR for each confusion matrix (TPR)
- Calculate fraction of test set predicted as positive (x)
- Plot $Lift = \frac{TPR}{x}$ vs x. Lift represents the advantage of a classifier over random guessing.

EXPECTED VALUE FORMALISM

Can be used for risk or profit/utility (negative risk)

$$EP = p(1,1)\ell_{11} + p(0,1)\ell_{10} + p(0,0)\ell_{00} + p(1,0)\ell_{01}$$

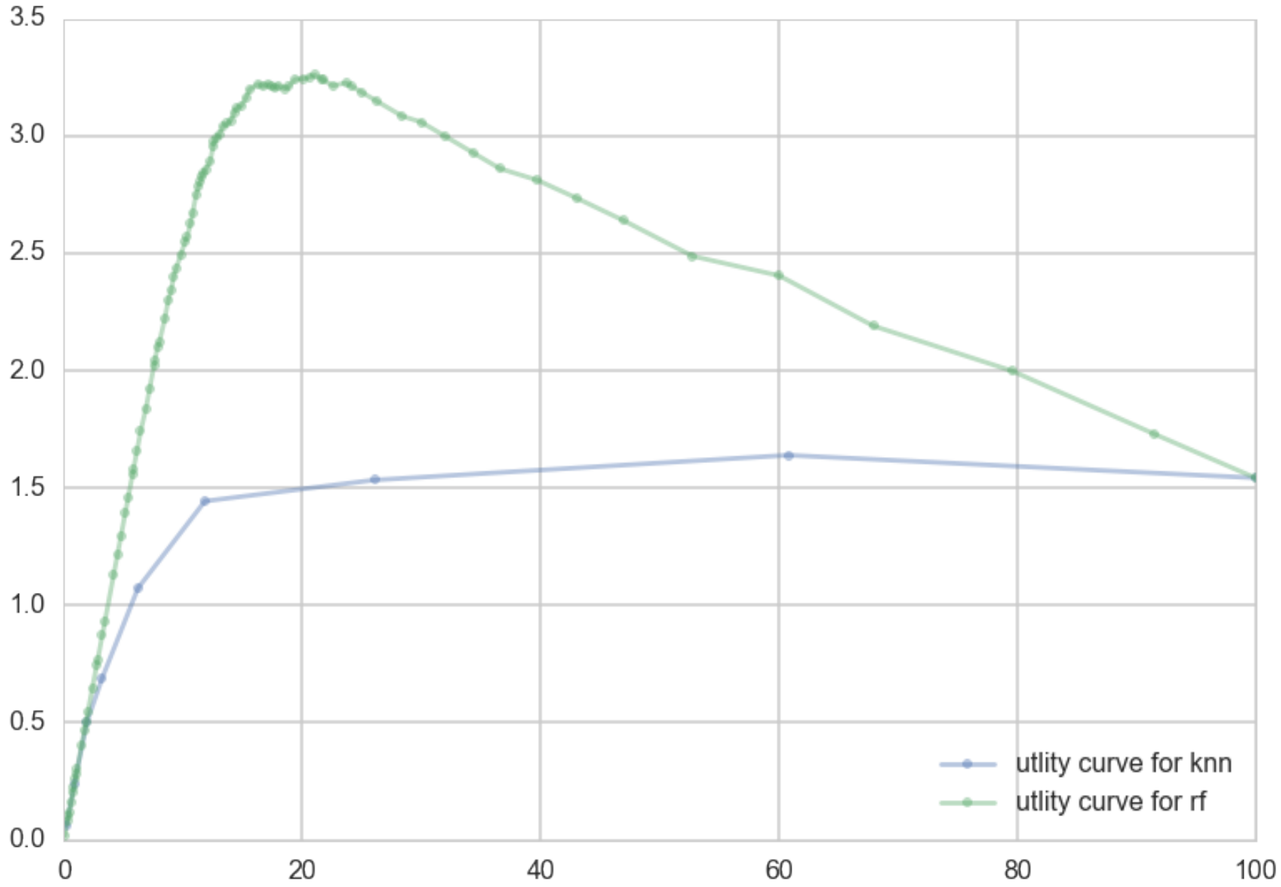
$$\begin{aligned} EP = p_a(1)[TPR\,\ell_{11} + (1 - TPR)\ell_{10}] \\ + p_a(0)[(1 - FPR)\ell_{00} + FPR\,\ell_{01}] \end{aligned}$$

Fraction of test set pred to be positive $x = PP/N$:

$$x = (TP + FP)/N = TPR\,p_o(1) + FPR\,p_o(0)$$

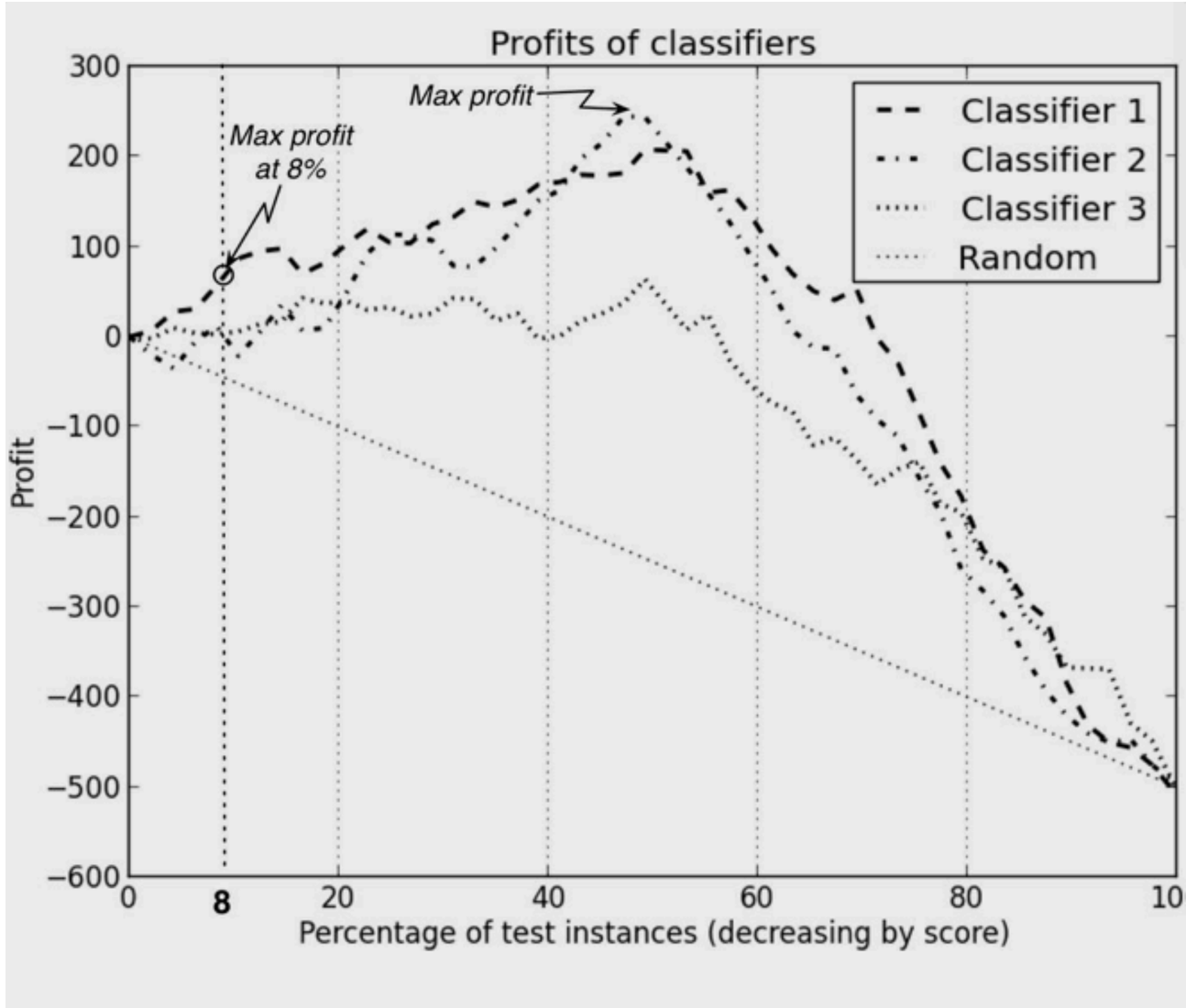
Profit curve

- Rank test set by prob/score from highest to lowest
- Calculate the expected profit/utility for each confusion matrix (U)
- Calculate fraction of test set predicted as positive (x)
- plot U against x



Finite budget[#]

- 100,000 customers, 40,000 budget, 5$ per customer
- we can target 8000 customers
- thus target top 8%
- classifier 1 does better there, even though classifier 2 makes max profit



[#] figure from Data Science for Business, Foster et. al.