Supervised Learning

Max Callaghan

2023-11-20

Introduction and Objectives

Assignment 4

Assignment 4 is due tomorrow night I am looking forward to receiving your submissions.

Objectives

Last week we asked if a text was **positive** or **negative** in sentiment.

This question is a special case of a common classification problem. Is a text X? Is it Y?

We will look at variants of this question and how to answer these by *training* a machine learning model to reproduce a given set of labels

Classification

(Re-)Introduction to classification

Classification is when we want to know if a given label or (labels) applies to a text.

Recall our first practical session where we built a rudimentary spam filter. We were looking for a functional form which allowed us to predict spam-ness as a function of our features.

```
from sklearn.feature_extraction.text import CountVectorizer
texts = [
    "Win cash", "free money", "free cash","cash money",
    "Thank you", "Best wishes","Cheers mate"
]
spam = [1,1,1,1,0,0,0]

vec = CountVectorizer()
dfmat = vec.fit_transform(texts)
pd.DataFrame(dfmat.todense(),columns=vec.get_feature_names_out(), index=texts)
```

```
##              best  cash  cheers  free  mate  money  thank  win  wishes  you
## Win cash        0     1       0     0     0      0      0    1       0    0
## free money      0     0       0     1     0      1      0    0       0    0
## free cash       0     1       0     1     0      0      0    0       0    0
## cash money      0     1       0     0     0      1      0    0       0    0
## Thank you       0     0       0     0     0      0      1    0       0    1
## Best wishes     1     0       0     0     0      0      0    0       1    0
## Cheers mate     0     0       1     0     1      0      0    0       0    0
```

(Re-)Introduction to classification

A very simple way of finding this functional form is via logistic regression

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(dfmat, spam)
```

```
## LogisticRegression()
```

```
print(clf.intercept_)
```

```
## [-0.01988335]
```

```
coefs = dict(zip(
  vec.get_feature_names_out(),
  clf.coef_.ravel().round(2)
))
coefs
```

```
## {'best': -0.33, 'cash': 0.73, 'cheers': -0.33, 'free': 0.5, 'mate': -0.33, 'money': 0.5, 'thank': -0.33, 'win':
```

(Re-)Introduction to classification

We can just add up the intercept and the coefficients, and apply a sigmoid function to this score .

$$t = \beta_0 + \beta_1 X_1 + \cdots + \beta_k X_k$$
$$p(X) = \sigma(t)$$

```
from scipy.stats import logistic

print(logistic.cdf(clf.intercept_ + coefs["cash"]))
```

```
## [0.67042693]
```

```
clf.predict_proba(vec.transform(["cash please"])).round(2)
```

```
## array([[0.33, 0.67]])
```

By *fitting* the logistic regression model, we have found a functional form which minimises a loss function (distance between predictions and reality)

Classification in a nutshell

Some would say we've done machine-learning again!

No matter how much fancier we get than logistic regression, the steps stay the same

1. Turn texts into features

Classification in a nutshell

Some would say we've done machine-learning again!

No matter how much fancier we get than logistic regression, the steps stay the same

1. Turn texts into features
2. Specify a model type

Classification in a nutshell

Some would say we've done machine-learning again!

No matter how much fancier we get than logistic regression, the steps stay the same

1. Turn texts into features
2. Specify a model type
3. *Fit* a model on our features and response value

Classification in a nutshell

Some would say we've done machine-learning again!

No matter how much fancier we get than logistic regression, the steps stay the same

1. Turn texts into features
2. Specify a model type
3. *Fit* a model on our features and response value
4. Using our fitted model, we can make predictions for any new text *in the same feature space*

Types of classification problems

So far we have explored simple binary classification tasks. In fact, we have three types

- **Binary classification**: Is a text spam OR not-spam
- **Multiclass classification**: Is a text written by Labour or the Conservatives or the Liberal Democrats (choose exactly one)
- **Multilabel classification**: Is a text about climate impacts or climate mitigation or climate adaptation (choose 0 or more).

The principle will remain the same, but we'll have to make a few changes to how we represent the problem. Pay particular attention to the difference between the last 2.

Measuring success

Training and validating

To evaluate a certain model, we train it on some data, and validate it on some **other** data.

To validate a model on **other** data means to use the features to make predictions for that data, then compare the predictions to the true values for our response variable.

It is **very** important that data does not **leak** from training to validation.

We want to understand not just how well a model fits the data it is trained on, but also how well this **generalises** to other similar data.

Metrics for classification tasks

Given a binary variable, when we compare the value of y with $\hat{y}$ (or `y_true` with `y_pred`), we can have 4 outcomes

- True positive (TP) - $y = 1 \quad \hat{y} = 1$
- True negative (TN) - $y = 0 \quad \hat{y} = 0$
- False positive (FP) - $y = 0 \quad \hat{y} = 1$
- False negative (FN) - $y = 1 \quad \hat{y} = 0$

A perfect classifier contains *only* **true positives** and **true negatives**.

Most validation metrics are various ways to count these up

Accuracy

Accuracy is simply the number of right decisions divided by the total number of decisions we make

$$\frac{TP + TN}{TP + TN + FP + FN}$$

It is bounded between 0 (every single decision was wrong), and 1 (every single decision was right).

It is the easiest metric to understand, but it can be misleading when data is unbalanced (which it mostly is)

Misleading Accuracy

Let's consider the case where 90% of emails are **not spam**. We can achieve an accuracy of 90% simply by always predicting 0.

```
y_true <- c(0,1,0,0,0,0,0,0,0,0)
y_pred <- c(0,0,0,0,0,0,0,0,0,0)
correct <- sum(y_true==y_pred)
accuracy <- correct/length(y_true)
accuracy
```

```
## [1] 0.9
```

Despite an impressive accuracy score, this classifier is not useful at all

Recall (sensitivity)

Recall (sometimes called sensitivity) measures something much more specific:

> *the proportion of truly positive samples we find by using our classifier*

Formally, this is given by

$$\frac{TP}{TP + FN}$$

```
tp <- sum((y_true==1)&(y_pred==1))
fn <- sum((y_true==1)&(y_pred==0))
recall <- tp / (tp+fn)
recall
```

```
## [1] 0
```

Precision

Precision, on the other hand, refers to

> *the proportion of positive predictions which are truly positive*

This is given by

$$\frac{TP}{TP + FP}$$

```
tp <- sum((y_true==1)&(y_pred==1))
fp <- sum((y_true==0)&(y_pred==1))
precision <- tp / (tp+fp)
precision
```

```
## [1] NaN
```

If we make zero positive predictions, we can't calculate precision.

F1 score

The F1 score is the harmonic (balanced) mean of precision and recall. It also ranges between 1 (perfect) and 0 (perfectly bad) but it is not so misleading given unbalanced data

Let's make a random 90% of our predictions correct, and flip the other 10%

```
accuracy_target <- 0.9

p = rep(1,100)
n = rep(0,900)
y <- sample(c(p,n))
y_pred <- y
perturb <- round(length(y)*(1-accuracy_target))
y_pred[1:perturb] <- 1 - y_pred[1:perturb]
tp <- sum((y==1)&(y_pred==1))
fn <- sum((y==1)&(y_pred==0))
fp <- sum((y==0)&(y_pred==1))
tn <- sum((y==0)&(y_pred==0))

recall <- tp / (tp+fn)
precision <- tp / (tp+fp)
f1 <- mean(c(recall,precision))
acc <- (tp+tn)/(tp+tn+fp+fn)
sprintf("Precision: %f, recall: %f, f1: %f, accuracy: %f", precision, recall, f1, acc)
```

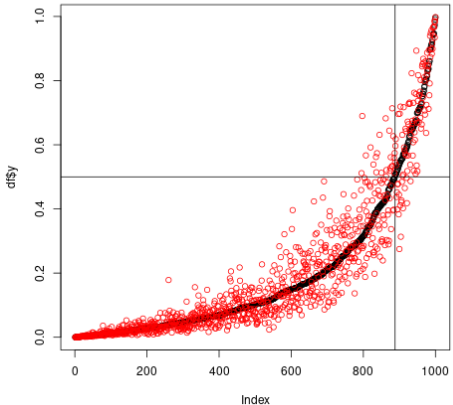
```
## [1] "Precision: 0.500000, recall: 0.910000, f1: 0.705000, accuracy: 0.900000"
```

Thresholds

The sigmoid function gives us score between 0 and 1. We usually treat 0.5 as a threshold, where everything above it belongs to the class and everything below it does not. However, we can adjust this threshold

```
library(stats)
library(dplyr)

n <- 1000
t <- rlogis(n, location=qlogis(0.1))
noise <- rnorm(n,sd=0.5)
y <- plogis(t)
y_pred <- plogis(t+noise)

df <- data.frame(y=y,y_pred=y_pred) %>%
  arrange(y)

png("plots/thresholds.png")
plot(df$y)
points(df$y_pred, col="red")
abline(v=which(df$y>0.5)[1])
abline(h=0.5)
dev.off()
```

```
## pdf
##   2
```



Precision-recall tradeoff

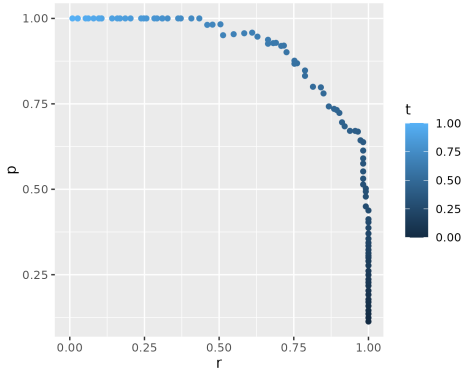
Adjusting the threshold therefore gives us different values of precision and recall

```
library(ggplot2)
df <- data.frame(t=numeric(), p=numeric(), r=numeric())
for (thresh in seq(0,1,0.01)) {
  tp <- sum((y>=0.5)&(y_pred>=thresh))
  fn <- sum((y>=0.5)&(y_pred<=thresh))
  fp <- sum((y<=0.5)&(y_pred>=thresh))
  recall <- tp / (tp+fn)
  precision <- tp / (tp+fp)
  df <- df %>% add_row(t=thresh, p=precision, r=recall)
}
ggplot(df, aes(r, p, colour=t)) +
  geom_point()
```

```
## Warning: Removed 1 rows containing missing values (`geom
```

```
ggsave("plots/precision-recall.png", width=5, height=4)
```

```
## Warning: Removed 1 rows containing missing values (`geom
```



Precision-recall tradeoff

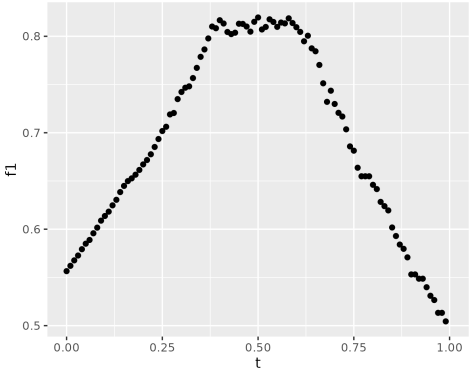
The maximum value of f1 does not necessarily lie at a threshold of 0.5 (be aware this is entirely synthetic data!)

```
library(ggplot2)

df$f1 <- (df$p + df$r) / 2

ggplot(df, aes(t, f1)) +
  geom_point()
```

```
## Warning: Removed 1 rows containing missing values (`geom
```

```
ggsave("plots/threshold-f1.png", width=5, height=4)
```

```
## Warning: Removed 1 rows containing missing values (`geom
```



Specificity

Specificity, also known as the **true negative rate** tells us:

What proportion of non-spam emails were correctly identified as not-spam

and it is given by

$$\frac{TN}{TN + FP}$$

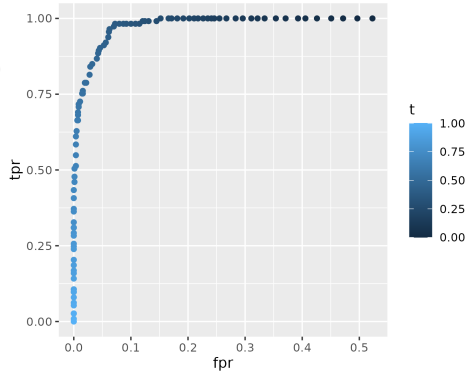
It's inverse, the **false positive rate**, tells us the probability of a false alarm.

ROC-AUC

Like precision, the false positive rate trades off against recall (sensitivity). The curve we get by examining this tradeoff at different thresholds is also called the Receiver operating characteristic (after its use in radar systems).

```
library(ggplot2)
df <- data.frame(t=numeric(), tpr=numeric(), fpr=numeric())
for (thresh in seq(0,1,0.01)) {
  tp <- sum((y>=0.5)&(y_pred>=thresh))
  fn <- sum((y>=0.5)&(y_pred<=thresh))
  fp <- sum((y<=0.5)&(y_pred>=thresh))
  tpr <- tp / (tp+fn)
  fpr <- fp / (fp+tn)
  df <- df %>% add_row(t=thresh, tpr=tpr, fpr=fpr)
}
ggplot(df, aes(fpr, tpr, colour=t)) +
  geom_point()
```

```
ggsave("plots/roc.png", width=5, height=4)
```



ROC-AUC

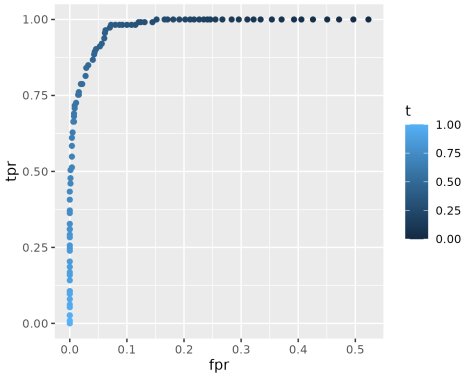
You can calculate the area under the curve, which is known as the ROC-AUC score

The ROC-AUC score tells us the probability that a randomly selected positive instance will be ranked higher than a randomly selected negative instance.

- ROC-AUC==1: the classifier is perfect
- ROC-AUC==0.5: the classifier is as good as random chance
- ROC-AUC<0.5: the classifier is worse than random chance

```
library(ModelMetrics)
auc(round(y),y_pred)
```

```
## [1] 0.9873891
```



Training, validation, and test splits

Any validation metric tells us how good a particular instance of a model performs in predicting the right labels for a particular set of documents it has not seen.

In practice, we want to do two things:

- Select a model which we think works best (by validating)
- Evaluate this best model on another set of data that has never been seen before

This gives us an estimate of how well our **best** model might perform on new data.

This means we **first** split our data into a training and a test split, and hold this test split back entirely. We must never use it to make decisions, but only to estimate performance of our chosen model.

We can further split our training set into further training and validation sets in order to make decisions about which model is best.

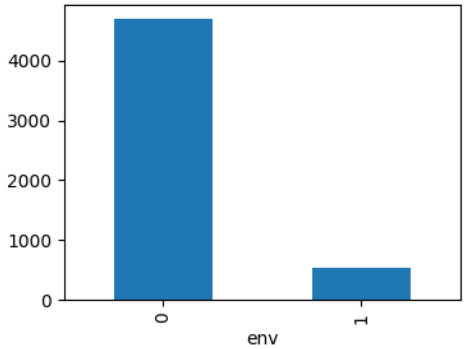
Training a model

Data

We're going to take our UK manifestos again, and use the annotations provided as labels. The label 501 is used to describe Environmental Protection source. We'll see if we can predict this label using machine learning.

As a first step, we can create a dummy variable for environmental protection, and look at how its distributed.

```
import pandas as pd
import numpy as np
fig, ax = plt.subplots(figsize=(4,3))
df = pd.read_csv("../datasets/uk_manifestos.csv")
df["env"] = np.where(df["cmp_code"]==501,1,0)
df.groupby(["env"])["text"].count().plot.bar(ax=ax)
plt.savefig(
  "plots/env_text_count.png", bbox_inches="tight"
)
```



Loading the data in R

```
library(readr)
library(tidyr)
library(dplyr)
df <- read_csv("../datasets/uk_manifestos.csv")
df<- df[sample(nrow(df)),]
df$env <- 0
df$env[df$cmp_code==501] <- 1
df$env <- factor(df$env)
```

Setting up a pipeline

Now we want to set up a pipeline that describes all the steps from data into predictions. This includes our usual vectorizer, then a Support Vector Machine classifier. These work by trying to fit a hyperplane that best separates the data in our multidimensional feature space.

```
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC
from sklearn.feature_extraction.text import TfidfVectorizer
clf = Pipeline(
    steps=[
        ("vect", TfidfVectorizer()),
        ("clf", SVC(probability=True, class_weight="balanced")),
    ]
)
```

Splitting train and test data

Now we split our data into train and test sets, and we can try fitting our pipeline on the training data. Once this is fit, we can make predictions on any new text data

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    df.text, df.env, test_size=0.2, random_state=42)

clf.fit(X_train, y_train)
```

```
## Pipeline(steps=[('vect', TfidfVectorizer()),
##                 ('clf', SVC(class_weight='balanced', probability=True))])
```

```
clf.predict_proba([
    "We will not raise taxes",
    "We will protect our natural resources"
])
```

```
## array([[0.98755764, 0.01244236],
##        [0.12685591, 0.87314409]])
```

Evaluating our model

Now we can evaluate the model on our test data. Although the accuracy is quite good, our F1 score is not wonderful.

```
from sklearn.metrics import f1_score, accuracy_score, precision_score, recall_score, roc_auc_score

y_pred = clf.predict_proba(X_test)

roc_auc = roc_auc_score(y_test, y_pred[:,1])
recall = recall_score(y_test, y_pred[:,1].round())
prec = precision_score(y_test, y_pred[:,1].round())
acc = accuracy_score(y_test, y_pred[:,1].round())
f1 = f1_score(y_test, y_pred[:,1].round())

print(f"ROC-AUC: {roc_auc:.1%}, Accuracy: {acc:.1%}, Precision: {prec:.1%}, recall: {recall:.1%}, F1 score: {f1:.1%
```

```
## ROC-AUC: 93.8%, Accuracy: 92.8%, Precision: 81.2%, recall: 52.0%, F1 score: 63.4%
```

Writing a Rrecipe

In the tidymodels ecosystem in R, a pipeline is called a "recipe"/"workflow"

```
library(tidymodels)
library(textrecipes)

df_split <- initial_split(df, prop=0.8)
train_data <- training(df_split)
test_data <- testing(df_split)

rec <-recipe(env ~ text, data = train_data) %>%
  step_tokenize(text) %>%
  step_tokenfilter(text, max_tokens = 1e3) %>%
  step_tfidf(text)

model <- svm_linear(mode="classification")

wf <- workflow() %>%
  add_recipe(rec) %>%
  add_model(model)

model_fit <- wf %>%
  fit(train_data)
```

Evaluating our model

Now we can evaluate the model on our test data. Although the accuracy is quite good, our F1 score is not wonderful.

```
test_data$prediction <- predict(model_fit, test_data)$.pred_class

scorer <- metric_set(
  yardstick::accuracy,
  yardstick::precision,
  yardstick::recall,
  yardstick::f_meas
)

scorer(test_data, truth=env, estimate=prediction, event_level="second")
```

```
## # A tibble: 4 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 accuracy  binary         0.918
## 2 precision binary         0.627
## 3 recall    binary         0.408
## 4 f_meas    binary         0.494
```

Optimizing and selecting a model

Tuning hyperparamters

There are a lot of different choices we can make in how we set up our model, and we can try out the effect of each of these, across 5 further splits of the training data. This takes quite a bit of time, so the set of parameter choices in the example is very small

```
from sklearn.model_selection import GridSearchCV

parameters = [
    {
        'vect__max_df': (0.5,),
        'vect__min_df': (5,),
        #'vect__ngram_range': ((1, 1), (1, 2)),
        'clf__kernel': ['linear'],
        #'clf__C': [1, 1e2]
    }
]

grid_search = GridSearchCV(
  clf, parameters, scoring="f1", n_jobs=4, verbose=1, cv=2
)
grid_search.fit(X_train, y_train)
```

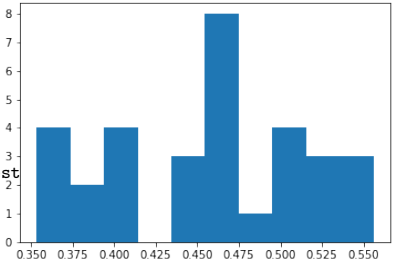
```
## Fitting 2 folds for each of 1 candidates, totalling 2 fits
## GridSearchCV(cv=2,
##              estimator=Pipeline(steps=[('vect', TfidfVectorizer()),
##                                        ('clf',
##                                         SVC(class_weight='balanced'
```

The spread of performance across hyperparameters

The results of each model specification are stored in .cv_results_ - we can quickly look at the spread of f1 scores across specifications

```
res = (pd.DataFrame(grid_search.cv_results_)
  .sort_values("rank_test_score",ascending=True)
)
hist = plt.hist(res["mean_test_score"])
res.head()
# The plot on the right -> comes from a wider
# search we carried out in the notebook
```

```
##    mean_fit_time  std_fit_time  ...  std_test_score  rank_test
## 0       1.034389      0.018585  ...        0.009588
##
## [1 rows x 13 columns]
```



Testing tuned hyperparameters

Now we can test the model specification that performed best in our tuning procedure on our test dataset (which it has *not* seen before)

```
y_pred = grid_search.predict_proba(X_test)

roc_auc = roc_auc_score(y_test, y_pred[:,1])
recall = recall_score(y_test, y_pred[:,1].round())
prec = precision_score(y_test, y_pred[:,1].round())
acc = accuracy_score(y_test, y_pred[:,1].round())
f1 = f1_score(y_test, y_pred[:,1].round())

print(f"ROC-AUC: {roc_auc:.1%}, Accuracy: {acc:.1%}, Precision: {prec:.1%}, recall: {recall:.1%}, F1 score: {f1:.1%
```

```
## ROC-AUC: 92.3%, Accuracy: 91.6%, Precision: 77.6%, recall: 41.6%, F1 score: 54.2%
```

Tuning hyperparameters in R

We can tune hyperparaters using the tune package in R. We say what parameters we want to tune, add this to a workflow, define the folds, and then apply `tune_grid()` to our workflow

```
model <- svm_poly(cost=tune()) %>%
  set_engine("kernlab") %>%
  set_mode("classification")

wf <- workflow() %>%
  add_recipe(rec) %>%
  add_model(model)

folds <- vfold_cv(train_data, v = 2)
svm_res <- tune_grid(
  wf, resamples = folds, grid = 2,
  metrics = metric_set(f_meas),
  control = control_grid(event_level="second")
  )
collect_metrics(svm_res)
```

```
## # A tibble: 2 x 7
##     cost .metric .estimator  mean     n std_err .config
##    <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 2.03   f_meas  binary     0.443     2  0.0277 Preprocessor1_Model1
## 2 0.0360 f_meas  binary     0.451     2  0.0421 Preprocessor1_Model2
```

Testing our best model

We then select our best model, and add finalise our workflow with this model

```
best_model <- svm_res %>% select_best()

final_workflow <- wf %>%
  finalize_workflow(best_model)

final_model <- final_workflow %>%
  fit(train_data)
```

```
##  Setting default kernel parameters
```

```
test_data$opt_prediction <- predict(final_model, test_data)$.pred_class
scorer(test_data, truth=env, estimate=opt_prediction, event_level="second")
```

```
## # A tibble: 4 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 accuracy  binary         0.893
## 2 precision binary         0.475
## 3 recall    binary         0.538
## 4 f_meas    binary         0.504
```

Threshold tuning

The fact that there is a large spread between precision and recall, and that ROC-AUC is much better than F1, indicates we may have a less than optimal threshold. Across 5 train-test splits of our train data, we can try out different thresholds using our best model.

```
X_train = X_train.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)

res = []

from sklearn.model_selection import KFold
kf = KFold(n_splits=5, random_state=None, shuffle=False)
k = 0
for train_index, test_index in kf.split(X_train):
    clf = grid_search.best_estimator_
    clf.fit(X_train[train_index], y_train[train_index])
    y_pred_k = clf.predict_proba(X_train[test_index])
    for t in np.linspace(0.1, 0.9, 50):
        y_pred_bin = np.where(y_pred_k[:,1]>t,1,0)
        res.append({
            "t": t,
            "f1": f1_score(y_train[test_index], y_pred_bin),
            "k": k
        })
    k+=1
```

Threshold tuning

```
optimal_t = res.groupby("t")["f1"].mean().sort_values(ascending=False).index[0]

print(optimal_t)
```

```
## 0.3122448979591837
```

```
y_pred_bin = np.where(y_pred[:,1]>optimal_t,1,0)
recall = recall_score(y_test, y_pred_bin)
prec = precision_score(y_test, y_pred_bin)
acc = accuracy_score(y_test, y_pred_bin)
f1 = f1_score(y_test, y_pred_bin)

print(f"Accuracy: {acc:.1%}, Precision: {prec:.1%}, recall: {recall:.1%}, F1 score: {f1:.1%}")
```

```
## Accuracy: 92.2%, Precision: 70.1%, recall: 60.0%, F1 score: 64.7%
```

Threshold tuning in R

The fact that there is a large spread between precision and recall, and that ROC-AUC is much better than F1, indicates we may have a less than optimal threshold. Across 5 train-test splits of our train data, we can try out different thresholds using our best model.

```
res <- data.frame(k=numeric(),t=numeric(),f1=numeric())
n_splits=5
folds <- vfold_cv(train_data, v = n_splits)
for (k in 1:n_splits) {
  k_split <- folds$splits[[k]]
  k_train <- training(k_split)
  k_test <- testing(k_split)
  k_model <- final_workflow %>% fit(k_train)
  k_test$pred <- predict(k_model, k_test, type="prob")$.pred_1
  for (t in seq(0.1, 0.9, length.out=50)) {
    est <- factor(ifelse(k_test$pred>=t,1,0))
    f1 <- f_meas_vec(k_test$env, est, event_level = "second")
    res <- add_row(res,k=k,t=t,f1=f1)
  }
}
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

Threshold tuning in R

```
res <- res %>%
  group_by(t) %>% summarise(f1 = mean(f1)) %>%
  arrange(desc(f1))

optimal_t <- res$t[1]

print(optimal_t)
```

```
## [1] 0.1816327
```

```
test_data$p <- predict(final_model, test_data, type="prob")$.pred_1
test_data$p_tuned <- factor(ifelse(test_data$p>=optimal_t,1,0))
scorer(test_data, truth=env, estimate=p_tuned, event_level="second")
```

```
## # A tibble: 4 x 3
##   .metric   .estimator .estimate
##   <chr>     <chr>          <dbl>
## 1 accuracy  binary         0.891
## 2 precision binary         0.469
## 3 recall    binary         0.575
## 4 f_meas    binary         0.517
```

Exercise

Choose a different category (or set of categories) from manifesto project data.

Build a classifier for your chosen category(ies).

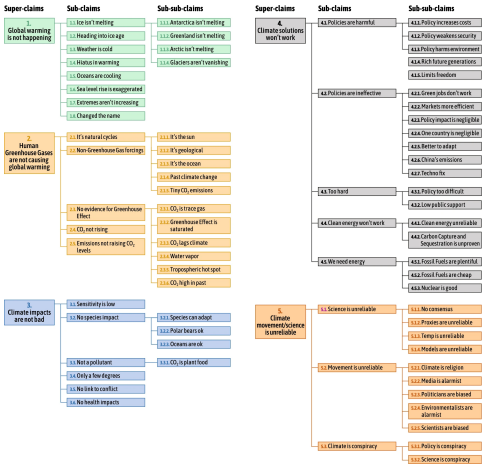
Report metrics on your classifier performance.

Investigate examples where the predictions are wrong. What could account for these?

Examples

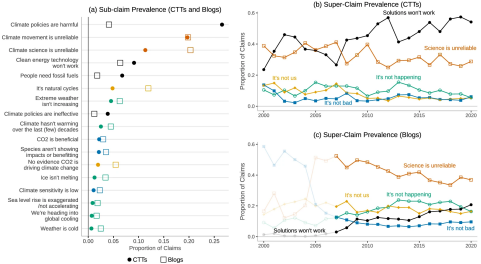
Computer-assisted classification of contrarian claims about climate change

Travis G. Coan and coauthors (2021) develop a machine-learning model which can classify contrarian claims about climate change according to a pre-defined taxonomy



Computer-assisted classification of contrarian claims about climate change

The use this to show how the evolution of climate denial in conservative think tanks (CTTs) and blogs, showing empirically that CTTs have shifted from denying that climate change is happening to arguing that solutions won't work



Computer-assisted classification of contrarian claims about climate change

They coded 31,000 paragraphs by hand to achieve these results. Best results were achieved with a combination of fancy (RoBERTa) and simply (Logistic) classifiers.

	Validation set (noisy)			Test set (noise free)		
	Precision	Recall	F1	Precision	Recall	F1
Logistic (unweighted)	0.71	0.55	0.62	0.83	0.57	0.68
Logistic (weighted)	0.62	0.68	0.65	0.75	0.70	0.72
SVM (unweighted)	0.66	0.56	0.61	0.77	0.58	0.66
SVM (weighted)	0.60	0.68	0.64	0.74	0.70	0.72
ULMFiT	0.69	0.69	0.69	0.77	0.67	0.72
ULMFiT (weighted)	0.66	0.60	0.62	0.76	0.60	0.65
ULMFiT (over sample)	0.41	0.73	0.50	0.46	0.75	0.55
ULMFiT (focal Loss)	0.66	0.58	0.60	0.73	0.56	0.61
ULMFiT-logistic	0.71	0.70	0.70	0.77	0.72	0.75
ULMFiT-SVM	0.74	0.65	0.70	0.81	0.63	0.71
RoBERTa	0.75	0.77	0.76	0.82	0.75	0.77
RoBERTa-logistic	0.76	0.77	0.76	0.83	0.75	0.79

Computer-assisted classification of contrarian claims about climate change

The performance varies somewhat across categories, but is better for the most coarse-grained categories

Code		Claim label	Precision	Recall	F1
0	0.0	**No claim**	0.90	0.95	0.93
1		**Global warming is not happening**	0.92	0.80	0.86
	1.1	Ice/permafrost/snow cover isn't melting	0.92	0.69	0.79
	1.2	We're heading into an ice age/global cooling	0.73	0.76	0.74
	1.3	Weather is cold/snowing	0.88	0.73	0.80
	1.4	Climate hasn't warmed/changed over the last (few) decade(s)	0.84	0.67	0.74
	1.6	Sea level rise is exaggerated/not accelerating	0.88	0.92	0.91
	1.7	Extreme weather isn't increasing/has happened before/isn't linked to climate change	0.93	0.86	0.90
2		**Human greenhouse gases are not causing climate change**	0.82	0.88	0.85
	2.1	It's natural cycles/variation	0.82	0.86	0.84
	2.3	There's no evidence for greenhouse effect/carbon dioxide driving climate change	0.69	0.79	0.73
3		**Climate impacts/global warming is beneficial/not bad**	0.91	0.92	0.91
	3.1	Climate sensitivity is low/negative feedbacks reduce warming	0.82	0.85	0.83
	3.2	Species/plants/reefs aren't showing climate impacts/are benefiting from climate change	0.81	0.90	0.85
	3.3	CO2 is beneficial/not a pollutant	0.90	0.96	0.93
4		**Climate solutions won't work**	0.86	0.64	0.74
	4.1	Climate policies (mitigation or adaptation) are harmful	0.70	0.55	0.61
	4.2	Climate policies are ineffective/flawed	0.88	0.44	0.59
	4.4	Clean energy technology/biofuels won't work	0.72	0.72	0.72
	4.5	People need energy (e.g. from fossil fuels/nuclear)	0.78	0.50	0.61
5		**Climate movement/science is unreliable**	0.82	0.75	0.78
	5.1	Climate-related science is unreliable/uncertain/unsound (data, methods & models)	0.77	0.80	0.77
	5.2	Climate movement is unreliable/alarmist/corrupt	0.78	0.61	0.69

Wrapup and Outlook

Wrapup

We've now covered almost everything in the course

We know how to represent texts, and we know how to use various methods to ask questions about the texts.

We've had a brief introduction to machine-learning, including how to train a model to reproduce *any* label we might apply to texts.

We've investigated how these methods work, learned how to be skeptical about what they can and can't represent, and we've had a look at a few examples of how they are used in research.

Outlook

In the last session we will look at how we might achieve some of these tasks (and some more tasks) using the very latest techniques from NLP research.

This will be Python-only!