Classification

Wojciech Wideł

Context and contents

- Target variable is **qualitative/categorical**
- An approach often taken is to model **the probability of an observation belonging to a category**
- A hyperparameter common for different methods is the **classification threshold**; if predicted probability of belonging to class c exceeds the threshold, then indeed assign class c to the observation

Today:

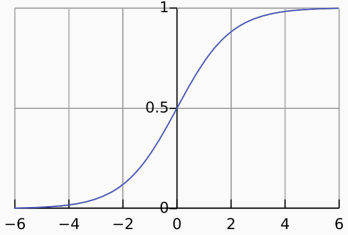
- Methods: logistic regression, linear discriminant analysis, K-nearest neighbours, support vector machines
- Accuracy metrics and when to use which
- Potential issues

Logistic regression: modelling probability

- Given $x \in \mathbb{R}^p$, how to predict probability $p(x) \in [0, 1]$ of something occurring?
- We know linear models, $x\beta$; since $x\beta \in \mathbb{R}$, one idea is to use a linear model for predicting some transformation of p.
- A meaningful transformation: $p \mapsto \frac{p}{1-p} \in [0, +\infty]$, **odds ratio**.
- Add one more step to get to the real line: $p \mapsto \log\left(\frac{p}{1-p}\right) \in \mathbb{R}$.
- The linear model $\widehat{\log\left(\frac{p}{1-p}\right)} = x\beta$ is equivalent to the non-linear model $\widehat{p} = \frac{1}{1+e^{-x\beta}}$.

Logistic regression for binary classification

- Two classes, $y \in \{0, 1\}$, goal: predict $p(y = 1|x)$, $x \in \mathbb{R}^p$
- Tool: logistic function $g(z) := \frac{1}{1+e^{-z}}$



- Properties:
  - $g(z) \in (0, 0.5)$ for $z < 0$, $\lim_{z \to -\infty} g(z) = 0$
  - $g(z) \in [0.5, 1)$ for $z \geq 0$, $\lim_{z \to \infty} g(z) = 1$

Logistic regression for binary classification

- Model: $\hat{p}(y_i = 1|x_i) = \frac{1}{1+e^{-x_i\beta}}$; will use $\mathbf{p(y_i)}$ for simplicity
- It follows that the **decision boundary is linear** in the inputs space:
  - $x_i\beta < 0$ implies $p(y_i) \in (0, 0.5)$,
  - $x_i\beta \geq 0$ implies $p(y_i) \in [0.5, 1)$.
- Data likelihood under the model:

$$l(\beta) = \left[\prod_{i:\ y_i=1} p(y_i)\right] \cdot \left[\prod_{i:\ y_i=0} (1 - p(y_i))\right]$$

- Fitted parameters (ideally): $\widehat{\beta}$ maximizing $l(\beta)$

The log loss function

$$
\begin{aligned}
\operatorname{argmax} l(\beta) = \operatorname{argmin} -l(\beta) &= \operatorname{argmin} - \left[\prod_{i\colon y_i=1} p(y_i) \right] \cdot \left[\prod_{i\colon y_i=0} (1 - p(y_i)) \right] \\
&= \operatorname{argmin} - \left[\sum_{i\colon y_i=1} \log p(y_i) \right] - \left[\sum_{i\colon y_i=0} \log(1 - p(y_i)) \right] \\
&= \operatorname{argmin} - \left[\sum_{i=1}^{n} y_i \log p(y_i) \right] - \left[\sum_{i=1}^{n} (1 - y_i) \log(1 - p(y_i)) \right] \\
&= \operatorname{argmin} - \sum_{i=1}^{n} \left[y_i \log p(y_i) + (1 - y_i) \log(1 - p(y_i)) \right]
\end{aligned}
$$

Thus, $\widehat{\beta}$ can be obtained by minimizing the last expression, known as the **log loss function**. This function is convex, and so gradient descent can be applied. Of course, it is a good idea to add a regularization term.

Multinomial logistic regression

- $y \in \{1, 2, \ldots, C\}$, goal: predict $p(y = c|x)$ for $c \in \{1, 2, \ldots, C\}$
- Tools: score functions $f(\beta, i) = e^{\beta x_i}$ and the softmax function $softmax(k, a_1, a_2, \ldots, a_n) := e^{a_k} / \sum_{i=1}^{n} e^{a_i}$
- Model: $\hat{p}(y_i = c|x_i) = softmax(c, \beta_1 x_i, \ldots, \beta_C x_i) = f(\beta_c, i) / \sum_k f(\beta_k, i)$
- Intuition: the greater $\beta_c x_i$ is, the more likely it is that x_i belongs to class c. Exponentiation of the values $\beta_k x_i$ exaggerates the differences between them.
- Data likelihood under the model is $l(\beta) = \prod_{i=1}^{n} \left(\prod_{c=1}^{C} \hat{p}(y_i = c|x_i)^{\delta_{c,y_i}} \right)$, where $\delta_{c,y_i} = 1$ if $c = y_i$ and 0 otherwise (the Kronecker delta).
- The negative log-likelihood is in this case the **cross-entropy**: $-\log l(\beta) = -\sum_{i=1}^{n} \sum_{c=1}^{C} \delta_{c,y_i} \log(\hat{p}(y_i = c|x_i))$.

Linear discriminant analysis

This method relies on Bayes theorem, which applied in the classification context is

$$p(y = c|X = x) = \frac{p(X = x|y = c) \cdot p(y = c)}{p(X = x)}.$$

Assuming that there are C classes, with n_c observations in class c, and that it is possible to estimate distributions of X in particular classes, to fill-in the right hand side of the equation we can use

- $\hat{p}(y = c) := \frac{n_c}{n}$, and
- $\hat{p}(X = x|y = c)$ – derived from data,

obtaining

$$p(y = c|X = x) = \frac{\hat{p}(X = x|y = c) \cdot n_c/n}{\sum_{l=1}^{C} \hat{p}(X = x|y = l) \cdot n_l/n}.$$

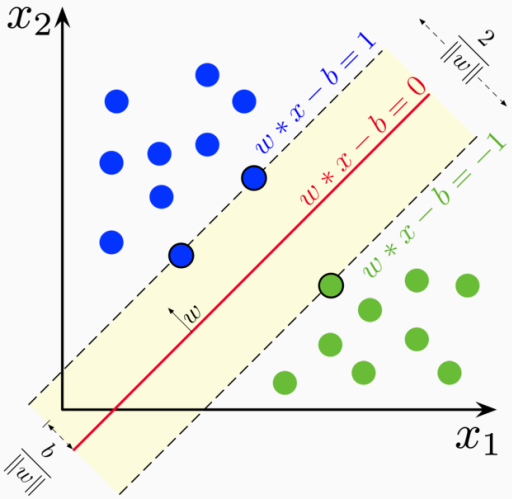
K-nearest neighbours

The probability of x belonging to class c is computed as

$$p(y = c|X = x) = \frac{1}{k} \sum_{x_i \in N_k(x)} \mathbb{1}_{y_i = c},$$

where $N_k(x)$ denotes the set of k training observations closest to x with respect to some metric.

Support vector machines: linearly separable case



- $\mathbf{y = 1}$, $\mathbf{y = -1}$
- $\mathbf{w^T x = b}$ – separating hyperplane
- With normalized or standardized data, there exist two parallel separating hyperplanes $\mathbf{w^T x - b = 1}$ and $\mathbf{w^T x - b = -1}$ with maximal possible distance between two separating hyperplanes
- The region bounded by them is called the **margin**
- The **maximum-margin hyperplane** is the hyperplane lying halfway between them

Support vector machines: finding the maximum-margin hyperplane

- Minimize $||w||$ (the distance between $\mathbf{w^T x - b = 1}$ and $\mathbf{w^T x - b = -1}$ is $2/||w||$).
- Separate the two groups: w and b must satisfy $w^T x_i - b \geq 1$ if $y_i = 1$, and $w^T x_i - b \leq -1$ if $y_i = -1$. Equivalently,

$$y_i(w^T x_i - b) \geq 1 \text{ for } i \in \{1, \ldots, n\}.$$

- Thus, we arrive at a quadratic programming problem:

$$\begin{aligned} &\text{minimize:} && ||w||_2^2 \\ &\text{subject to:} && y_i(w^T x_i - b) \geq 1, \quad i \in \{1, \ldots, n\} \\ & && b \in \mathbb{R}, w \in \mathbb{R}^p \end{aligned}$$

Closing remarks on linearly separable case.

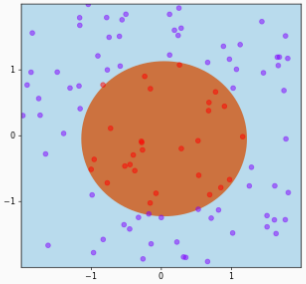
- The maximum-margin hyperplane is determined by the points lying the closest to it. They are called **support vectors**.
- The corresponding maximal margin classifier is $x \mapsto \textbf{sgn}(w^T x_i - b)$.

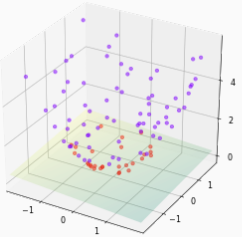
Modification for non-separable case: allow misclassifications, but penalize them.

$$\begin{aligned} \text{minimize:} \quad & ||w||_2^2 + C \sum_{i=1}^{n} \zeta_i \\ \text{subject to:} \quad & y_i(w^T x_i - b) \geq 1 - \zeta_i, \qquad i \in \{1, \ldots, n\} \\ & \zeta_i \geq 0, b \in \mathbb{R}, w \in \mathbb{R}^p \end{aligned}$$

Support vector machines: non-linear separation

- Idea 1: apply a non-linear transformation ϕ to the inputs that will map them on a space of a higher dimension. Fit SVM in that space.





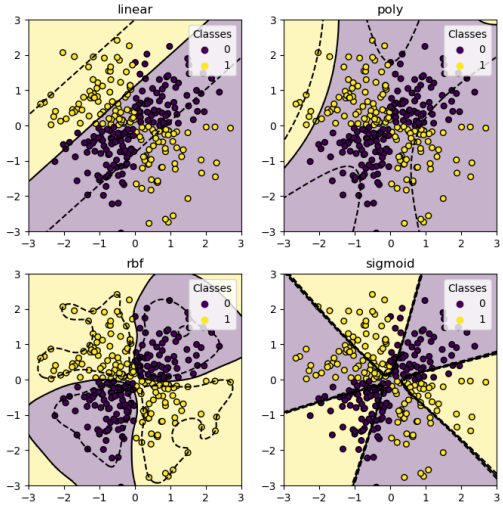
- Idea 2: sufficient to know $\phi(x)^T\phi(x') =: K(x, x')$, called **the kernel**, since inputs impact the fitting process solely via the dot product.

Support vector machines: examples of kernels

- Polynomial, homogeneous: $K(x, x') = (x^T x')^d$
- Polynomial, inhomogeneous: $K(x, x') = (x^T x' + r)^d$
- Gaussian radial basis function (RBF): $K(x, x') = \exp\left(-\gamma ||x - x'||^2\right)$ for $\gamma > 0$
- Sigmoid function: $K(x, x') = \tanh\left(\kappa x^T x' + c\right)$

Remark. The non-linear transformation trick could also be applied for other classifiers with linear decision boundary (e.g., logistic regression). The drawbacks are (1) the need for explicit transformation definition, (2) increased complexity of the parameters fitting process due to huge increase in the input space dimension.

Support vector machines: illustration



Support vector machines: multinomial case

Fit separate SVM classifier for every class $c \in \{1, 2, \ldots, C\}$, with each of them fitted for the binary classification $y = c$ vs $y \neq c$.

Remark. The same approach (one-vs-rest) can be used for logistic regression and any other classification method.

Confusion matrix and accuracy metrics

		Observed class: Positive	Observed class: Negative	Total
Predicted class	Positive	TP	FP	$TP + FP$
	Negative	FN	TN	$FN + TN$
	Total	$TP + FN$	$FP + TN$	n

- Accuracy: (TP+TN)/n
- True positive rate/sensitivity/recall: TP/P = TP/(TP+FN)
- False positive rate/specificity: FP/N = FP/(FP+TN)
- Precision: TP/(TP+FP)
- F1-score: 2(PREC·RECALL)/(PREC+RECALL)

When to use which accuracy metric

Remark. Hardly ever only one metric is used.

- Accuracy: when the classes are balanced and each class is equally important
- True positive rate: when identifying positives is crucial, even at the cost of raising false alarms (e.g., identifying fraudulent transactions)
- False positive rate: when the cost of raising an alert is costly
- Precision: when the follow-up actions for positive identification are costly
- F1-score: it is the harmonic mean of precision and recall, optimizing it strikes balance between the two

Potential issues: imbalanced classes

Maybe 95% of your training emails are regular ones (**majority class**), and only 5% are spam (**minority class**).

- Some evaluation metrics will be misleading; e.g., model assigning majority class to every observation will achieve 95% accuracy on the training set.
- Model will be biased towards the majority class, as it learns more about it; i.e., it will perform better on the majority class.
- Model will generalize poorly to new data, especially for the minority class.
- Misclassifying observations from the minority class might have costly consequences (e.g., cancer diagnosis).

Potential fixes for imbalanced classes

- Choose proper evaluation metric. The F1 score might be a good choice.
- Resample your data, so that the sizes of the two classes are equal, e.g.,
  - under-sample majority class: remove randomly selected observations from the majority class,
  - over-sample minority class: add randomly selected copies of minority class observations,
  - use synthetic minority oversampling (SMOTE): create synthetic minority class observations.
- Penalize learning algorithm so that mistakes made on minority class contribute more to the loss function. For SVM in Python this can be done with `classifier = SVC(class_weight='balanced', probability=True)`.
- Try different algorithms. Tree ensembles (random forests, gradient boosted trees, etc.) are known to be effective in this case.

An example

TODO; plot scores vs classification threshold

Model selection/hyperparameters tuning via grid search

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.linear_model import Lasso, Ridge
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score

X = df[features]
y = df[target]
for scaler in [StandardScaler, MinMaxScaler]:
    for m in [Lasso, Ridge]:
      for alpha in np.linspace(0, 30, 1000):
        model = m(alpha=alpha)
        pipeline = Pipeline([("scaler", scaler()), ("model", model)])
        cv = KFold(n_splits=10)
        scores = cross_val_score(pipeline, X, y, cv=cv, scoring="neg_mean_squared_error")
        score = -np.mean(scores)
        # code for keeping track of scores omitted here
```

Model selection/hyperparameters tuning using hyperopt-sklearn

```
from hpsklearn import HyperoptEstimator, linear_regression, lasso, ridge , any_preprocessing
from hyperopt import tpe
from hyperopt import hp
from sklearn.metrics import mean_squared_error

X_train = df[features]
y_train = df[target]
reg_alpha = hp.loguniform("alpha", low=np.log(1e-5), high=np.log(50))
models = hp.choice("regressor",
        [linear_regression("lr"),
        lasso("lasso", alpha=reg_alpha),
        ridge("ridge", alpha=reg_alpha)])
estim = HyperoptEstimator(regressor=models, preprocessing=any_preprocessing("my_pre"),
            algo=tpe.suggest, max_evals=200,
            trial_timeout=120, loss_fn=mean_squared_error)
estim.fit(X_train, y_train, n_folds=5, cv_shuffle=True)
print(estim.best_model())
```

Summary

- Remember about classification threshold
- Logistic regression:
  - Models the probability of an observation belonging to one of the classes.
  - Decision boundary is linear in the inputs space.
  - Parameters fitted by maximizing data likelihood (minimizing the negative log-likelihood, log loss function).
- Support vector machines:
  - xoxo
  - xoxo
  - xoxo

References

- G. James, D. Witten, T. Hastie and R. Tibshirani, *An Introduction to Statistical Learning*
- https://en.wikipedia.org/wiki/Support_vector_machine
- https://neptune.ai/blog/evaluation-metrics-binary-classification