method_model_types

February 16, 2016

1 Three types of models

This notebook shows how different types of classifiers can be used to predict categorical (e.g. present or absent), ordinal (e.g. stimulus contrast) and circular data (e.g. stimulus angle).

For clarity purposes, we won't consider the cross validation and temporal loops here, but solely focus on how each estimator is built and scored.

Note that in the manuscript, the estimator are preceeded by a normalization step (`StandardScaler`).

2 Prepare Data

In [1]:
```
import numpy as np
from sklearn.svm import SVC, LinearSVR
from sklearn.metrics import roc_auc_score
from jr.gat import AngularRegression
```

In [2]:
```
n_trial = 100

def make_data(y, n_chan=20, snr=1):
    """Simulate `n_trials` measured at sensor level at a single time sample in a given subject.
    X = np.zeros((n_trial, n_chan))
    y = y[:, None] if y.ndim == 1 else y  # ensure n_trials x n_dims

    # Mixture of a neural source (one for each dimension of y) projected onto the MEG sensors
    for source in y.T:
        # Setup a random foward model (from source to sensors): ~ set a random source
        forward = np.random.randn(n_chan, 1)

        # Add projection to sensors for each trial
        X += np.dot(forward, source[:, None].T).T

    # Add common activity to all trials
    common_activity = np.random.randn(n_chan)
    X += np.tile(common_activity, [n_trial, 1])

    # Add background noise
    X += np.random.randn(*X.shape) / snr

    return X


def cv(X, y):
    """Divides into a single train and a test set"""
```

```
    # Note that in the manuscript we don't use a validation, but a cross validation.
    train = range(len(X)//2)
    test = range(len(X)//2, len(X))
    return (X[train], y[train]), (X[test], y[test])
```

3 Categorical model

```
In [3]: # Categorical data are fitted with a linear SVC that output probabilistic estimates, and scored
        y = np.random.randint(0, 2, n_trial)  # y is a list of 0 or 1.
        X = make_data(y)

        # Estimator
        clf = SVC(kernel='linear', probability=True)

        # Scorer
        def scorer(y_true, y_pred):
            """Score probabilistic outputs for a single class as the other class is the numerical compl
            return roc_auc_score(y_true, y_pred[:, 1])

        # Fit, predict, and score
        (X_train, y_train), (X_test, y_test) = cv(X, y)
        clf.fit(X_train, y_train)
        y_pred = clf.predict_proba(X_test)
        score = scorer(y_test, y_pred)  # score in [0, 1], chance = .5
        print('score:', score)  # should be > to .5

        # Try shuffling the data:
        np.random.shuffle(y)
        (X_train, y_train), (X_test, y_test) = cv(X, y)
        clf.fit(X_train, y_train)
        y_pred = clf.predict_proba(X_test)
        score = scorer(y_test, y_pred)  # score in [0, 1], chance = .5
        print('random score:', score)  # should be close to .5
```

```
('score:', 0.98357963875205257)
('random score:', 0.3814935064935065)
```

4 Ordinal model

```
In [4]: # Categorical data are fitted with a linear SVR, and scored with a spearman regression.
        y = np.random.rand(n_trial)  # y is list of random float values between 0 and 1.

        # Add information on a third of the channels so that `X` encodes `y`.
        X = make_data(y)

        # Estimator
        clf = LinearSVR()

        # Scorer: non parametric R^2
        def scorer(y_true, y_pred):
            from scipy.stats import spearmanr
            rho, p = spearmanr(y_true, y_pred)
            return rho
```

```
        # Fit, predict, and score
        (X_train, y_train), (X_test, y_test) = cv(X, y)
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        score = scorer(y_test, y_pred)  # score in [-1, 1], chance = 0.
        print('score:', score)  # should be > 0.

        # Try shuffling the data:
        np.random.shuffle(y)
        (X_train, y_train), (X_test, y_test) = cv(X, y)
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        score = scorer(y_test, y_pred)
        print('random score:', score)  # should be close to 0.
```

```
('score:', 0.50232893157262903)
('random score:', -0.02569027611044418)
```

5 Circular model

In [5]:
```
        # Circular data are fitted with two linear SVR, and scored with an angle error.
        y = 2 * np.pi * np.random.rand(n_trial)  # y is list of random float values between 0 and 2 * p

        # Add angular information through two neural sources respectively coding for the cos and the si
        y_cos_sin = np.vstack((np.cos(y), np.sin(y))).T

        # Obviously, this encoding scheme differs from our brain's.
        # However:
        # 1) This modelling approach is the simplest possible model for angular data.
        # 2) Myers, Edward, Stokes et al. eLife (2015) show evidence that Gabor orientations are encode
        #    with circular representations in the MEG signals (which makes sense, knowing the encoding
        # 3) This approach empirically works and thus validate our modeling hypothesis.

        X = make_data(y_cos_sin)

        # Estimator
        random_state = 42  # to show that the subsequent methods are equivalent
        clf_sin = LinearSVR(random_state=random_state)
        clf_cos = LinearSVR(random_state=random_state)

        # Scorer: angle error
        def scorer(y_true, y_pred):
            """Scoring function dedicated to AngularRegressor"""
            pi = np.pi
            angle_errors = y_true - y_pred
            score = np.mean(np.abs((angle_errors + pi) % (2 * pi) - pi))  # in [0, pi], chance = pi / 2

            # For vizualization clarity we actually report a score with a chance level at 0
            # and that has increasing values with increasing accuracy.
            score = np.pi / 2 - score  # in [-pi/2, pi/2], chance = 0
            return score
```

```
        # Fit, predict, and score
        # ---- cv
        (X_train, y_train), (X_test, y_test) = cv(X, y)
        # ---- fit
        clf_cos.fit(X_train, np.cos(y_train))
        clf_sin.fit(X_train, np.sin(y_train))
        # ---- predict
        y_pred = np.arctan2(clf_sin.predict(X_test),
                            clf_cos.predict(X_test))
        # ---- score
        score = scorer(y_test, y_pred)
        print('score:', score)


        # This angular estimator is defined in `jr.gat` as `AngularRegression` and directly takes circu
        # data as a `y` parameter.
        clf = AngularRegression(clf=LinearSVR(random_state=random_state))
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        print('score:',  scorer(y_test, y_pred))  # should be > 0

        # Try shuffling the data to check chance level
        np.random.shuffle(y)
        (X_train, y_train), (X_test, y_test) = cv(X, y)
        clf = AngularRegression(clf=LinearSVR(random_state=random_state))
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        print('random score:', scorer(y_test, y_pred))  # should be around 0
```

```
('score:', 1.3070900553202693)
('score:', 1.3070900553202693)
('random score:', -0.015077557641889783)
```