Supervised Machine Learning Cheat Sheet

You can also find the latest updates to this cheat sheet on my Git Hub repository.

The purpose of my writing is to provide some insight into the process of machine learning, especially for people who are new to this field. I am currently a data science intern at Cialfo and have enjoyed it so far, but sometimes I find myself feeling overwhelmed by the sheer number of supervised machine learning models available. So after a meticulous review of various sources, I have crafted a cheat sheet that even an individual with little to no experience in machine learning can use efficiently.

To begin with i have created a table that shows you when to use what model and their advantages and disadvantages.

> **Note :** There are a lot more models , i have only covered the important ones.

Model Name	Classification / Regression	Pros	Cons
Linear Regression	Regression	Simple Interpretable Scientifically accepted Widely available	Sensitive to outliers Data must be independent
Logistic Regression	Classification	Easier to implement Easier to interpret very efficient to train	Over fitting on high dimension data Non linear problems cant be solved Sensitive to outliers
KNN (K Nearest Neighbors)	Both	No Training Period New data can be added seamlessly Easy to implement	Large memory requirements Cannot process highly dimensional data

Model Name	Classification / Regression	Pros	Cons
Decision Trees	Both	Requires little data preparation Able to handle both numerical and categorical data Able to handle multi-output problems	Highly unstable Relatively innacurate
Random Forest	Both	Robust to outliers Lower risk of overfitting Better accuracy than other classification algorithms	Prone to over fitting Cannot guarantee opptimal trees Low accuracy
Gradient Boosted Tree	Both	provides predictive accuracy that cannot be trumped Flexibility	Over emphasize outliers Computationaly expensive
SVM (Support Vector Machines)	Both	High Flexibility Accuracy Works well with high dimensionality data	Not suitable for large data sets

After i choose the machine learning models i would find my self looking back at my past documentation or sci-kit learn help. This next section should help you resolve this problem with out having to go through several sources.

- **Linear Regression**

1. Create a train and test data set

```
msk = np.random.rand(len(df))< 0.8
train = cdf[msk]
test = cdf[~msk]
```

2. Model

```
from sklearn.linear_model import LinearRegression

lm = LinearRegression()

train_x = np.asanyarray(train[['insert independent variables']])

train_y = np.asanyarray(train[['insert target variable']])

lm.fit(x,y)
```

3. Prediction

```
yhat = lm.predict(test_x)
```

- **Decision Tree Regressor for Regression**

1. Instantiate dt

```
dt = DecisionTreeRegressor(max_depth=8,
             min_samples_leaf=0.13,
            random_state=3)
```

2. Fit dt to the training set

```
dt.fit(X_train, y_train)
```

3. Compute y_pred

```
y_pred = dt.predict(X_test)
```

- **Logistic Regression**

Some of you mistake logistic regression as regression model , it is used for classification.They predict binary values such as YES or NO , based on prior observations in a data set.

1. Create training and test sets

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.4, random_state=42)
```

2. Create Fit the classifier to the training data

```
logreg = LogisticRegression().fit(X_train,y_train)
```

3. Predict the labels of the test set: y_pred

```
y_pred = logreg.predict(X_test)
```

- **KNN (K Nearest Neighbors)**

1. Normalize - This is very important as KNN uses distance between points to classify

```
x = Preprocessing.StandardScaler().fit(x).transform(x.astype(float))
```

2. Create training and test sets

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.4, random_state=42)
```

3. Training the model

```
from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=4)
neigh.fit(x_train,y_train)
```

4. Predicting

```
y_pred = neigh.predict(x_test)
```

- **Decision Tree Classifier**

1. Ordinal Encoder - encodes categorical variables in the data

```
from category_encoders.ordinal import OrdinalEncoder

Ordinal = OrdinalEncoder()
encoded_data = Ordinal.fit_transform(df.drop(['insert target variable'], axis=1))
```

2. Create training and test sets

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.4, random_state=42)
```

3. Instantiate a DecisionTreeClassifier 'dt' with a maximum depth of 6

```
dt = DecisionTreeClassifier(max_depth=6, criterion ='gini' , random_state=SEED)
```

4. Fit dt to the training set

```
dt.fit(X_train, y_train)
```

5. Predict test set labels

```
y_pred = dt.predict(X_test)
print(y_pred[0:5])
```

- **Bagging Classifier**

Provides better accuracy than normal decision tree, bagging takes and trains on model with various subsets of the data set

it could use the same subset n times during the subset selection process

1. Import BaggingClassifier

```
from sklearn.ensemble import BaggingClassifier
```

2. Instantiate dt

```
dt = DecisionTreeClassifier(random_state=1)
```

3. Instantiate bc

```
bc = BaggingClassifier(base_estimator=dt, n_estimators=50, random_state=1)
```

- **Random Forest**

1. Instantiate rf

```
rf = RandomForestRegressor(n_estimators=25,
            random_state=2)
```

2. Fit rf to the training set

```
rf.fit(X_train, y_train)
```

3. Predict the test set labels

```
y_pred = rf.predict(X_test)
```

- **Gradient Boosting**

1. Import GradientBoostingRegressor

```
from sklearn.ensemble import GradientBoostingRegressor
```

2. Instantiate sgbr

```
sgbr = GradientBoostingRegressor(max_depth=4,
                                 subsample=0.9,
                                 max_features=0.75,
                                 n_estimators=200,
                                 random_state=2)

^^ max _ features = 0.75 meaning it uses 75% of the data to train
```

3. Fit sgbr to the training set

```
sgbr.fit(X_train, y_train)
```

4. Predict test set labels

```
y_pred = sgbr.predict(X_test)
```

- **ADA Boosting**

In ada boost each predictor pays more attention to the instances wrongly predicted by its predecessor.

1. Instantiate dt

```
dt = DecisionTreeClassifier(max_depth=2, random_state=1)

2. Instantiate ada
ada = AdaBoostClassifier(base_estimator=dt, n_estimators=180, random_state=1)

3. Fit ada to the training set
ada.fit(X_train, y_train)

3. Compute the probabilities of obtaining the positive class
y_pred_proba = ada.predict_proba(X_test)[:,1]
```