Les techniques standards appliquées à OpenSolarMap (1/3)

23/06/2016

Lorsqu'un algorithme simple ne convient pas, la deuxième étape d'un projet de machine learning est d'essayer des « grands classiques ». Ces algorithmes sont plus complexes d'un point de vue théorique, mais des implémentations toutes prêtes existent et cette étape est généralement rapide à mettre en œuvre.

Régression Logistique

La régression logistique porte un nom déroutant puisque cette méthode est utilisée autant pour des problèmes de régression que de classification. De plus, le choix par Pierre François Verhulst terme « logistique » est aujourd'hui un mystère. Pourtant, la régression logistique est sans doute la méthode la plus répandue pour traiter des problèmes de classification comme c'est le cas ici.

Il existe une multitude d'implémentations de la régression logistique. La méthode utilisée pour OpenSolarMap est celle de Scikit-Learn. Scikit-Learn est un ensemble d'implémentations en langage Python d'algorithmes courants. Cette librairie maintenue par l'INRIA est très populaire partout dans le monde. Voir la documentation de l'implémentation : http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

Entraîner puis tester un modèle de régression logistique requiert peu de code à écrire :

```
1 train_data, val_data, test_data = load.load_all_data(train_ids, val_ids, test_ids, l, colo
2 model = sklearn.linear_model.LogisticRegression(penalty='l2', C=1e10)
3 model.fit(train_data, train_labels)
4 predictions = model.predict(val_data)
5 err = (predictions != val_labels).sum() / len(val_labels)
```

Passons en revue chaque ligne :

1. Les données sont chargées dans les variables `train_data`, `val_data` et `test_data`. La fonction `load.load_all_data()`, spécifique a notre problème, prend en paramètre la liste des identifiants de toits à charger, la taille `l` des images voulue et le nombre de canaux de couleur voulus (rouge, vert et bleu ou noir et blanc). Les images de toitures sont séparées en 3 échantillons :
    1. Un échantillon d'apprentissage.
    2. Un échantillon de test.
    3. Un échantillon de validation.
2. Un objet python encapsulant un modèle de régression linéaire est créé. Les paramètres `penalty` et `C` configurent la régularisation. La régularisation est utile lorsque le nombre de features est comparable à la taille de l'échantillon. Ici, il y a plusieurs milliers d'exemples dans l'échantillon d'apprentissage et quelques centaines de features tout au plus. Pour simplifier, le paramètre `C` a une valeur très élevée (`1e10 = 10.000.000.000`) ce qui correspond à une régularisation négligeable.
3. Le modèle est entraîné sur l'échantillon d'apprentissage. Le modèle a accès aux features (`train_data`) mais aussi aux labels (`train_labels`) pour pouvoir se corriger et s'améliorer.
4. Le modèle fait des prédictions sur l'échantillon de test. Maintenant le modèle n'a pas accès aux labels.
5. Le taux d'erreurs de la prédiction du modèle est calculé comme le quotient du nombre d'erreurs sur la taille de l'échantillon.

On choisit la taille des images `l` et le choix de couleurs (rouge, vert et bleu ou noir et blanc) en essayant plusieurs combinaison. La figure 1 montre que la taille qui donne les meilleurs résultats est de 6 pixels par 6 pixels. Le fait de tester successivement plusieurs hyper-paramètres (les paramètres, comme `l`, qui sont extérieurs au modèle de régression logistique et définis par le data-scientist) peut provoquer un phénomène appelé surapprentissage. Il est nécessaire de valider la performance sur un échantillon qui n'a été utilisé ni durant l'apprentissage ni durant la phase de test, l'échantillon de validation. Dans notre situation, le taux d'erreur sur l'échantillon de validation est de 12.5%.

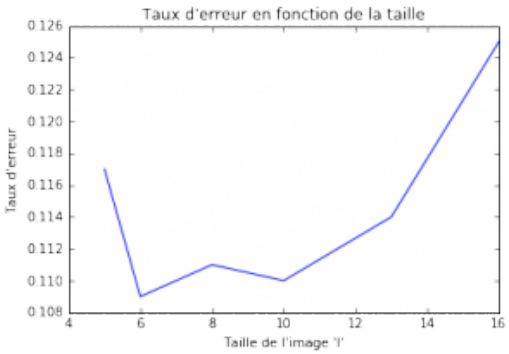


Figure 1 : choix des hyperparamètres pour la régression logistique

Support Vector Machines

Si la régression logistique a été développée dans la fin des années 60 par le statisticien David Cox et elle est maintenant considérée comme un outil de statistique classique, les « machines à vecteurs de support » sont développées depuis les années 90 et constituent encore un domaine de recherche très actif. Cette différence d'âge, ainsi que le fait que l'analyse mathématique de ces deux méthodes est très différente fait souvent oublier que les performances, tant en prédiction qu'en temps de calcul, sont souvent très semblables.

Passer de la régression logistique au Support Vector Classifier (SVC) est presque immédiat, il faut remplacer la ligne

```
model = sklearn.linear_model.LogisticRegression(penalty='l2', C=1e10)
```

par la ligne

```
model = sklearn.svm.LinearSVC(penalty='l2', C=1e10, dual=False)
```

Le paramètre `dual=False` commande à la librairie Scikit-Learn de ne pas utiliser l'implémentation « duale », qui est appropriée dans les cas où le nombre de features est plus important que la taille de l'échantillon.

Le meilleur résultat est toujours obtenu avec une taille de 6 pixels par 6 pixels, mais cette fois-ci en couleurs (rouge, vert et bleu). Le résultat de l'étape de la validation est aussi de 12.5%.

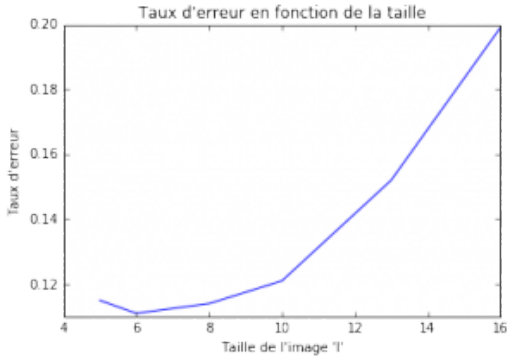
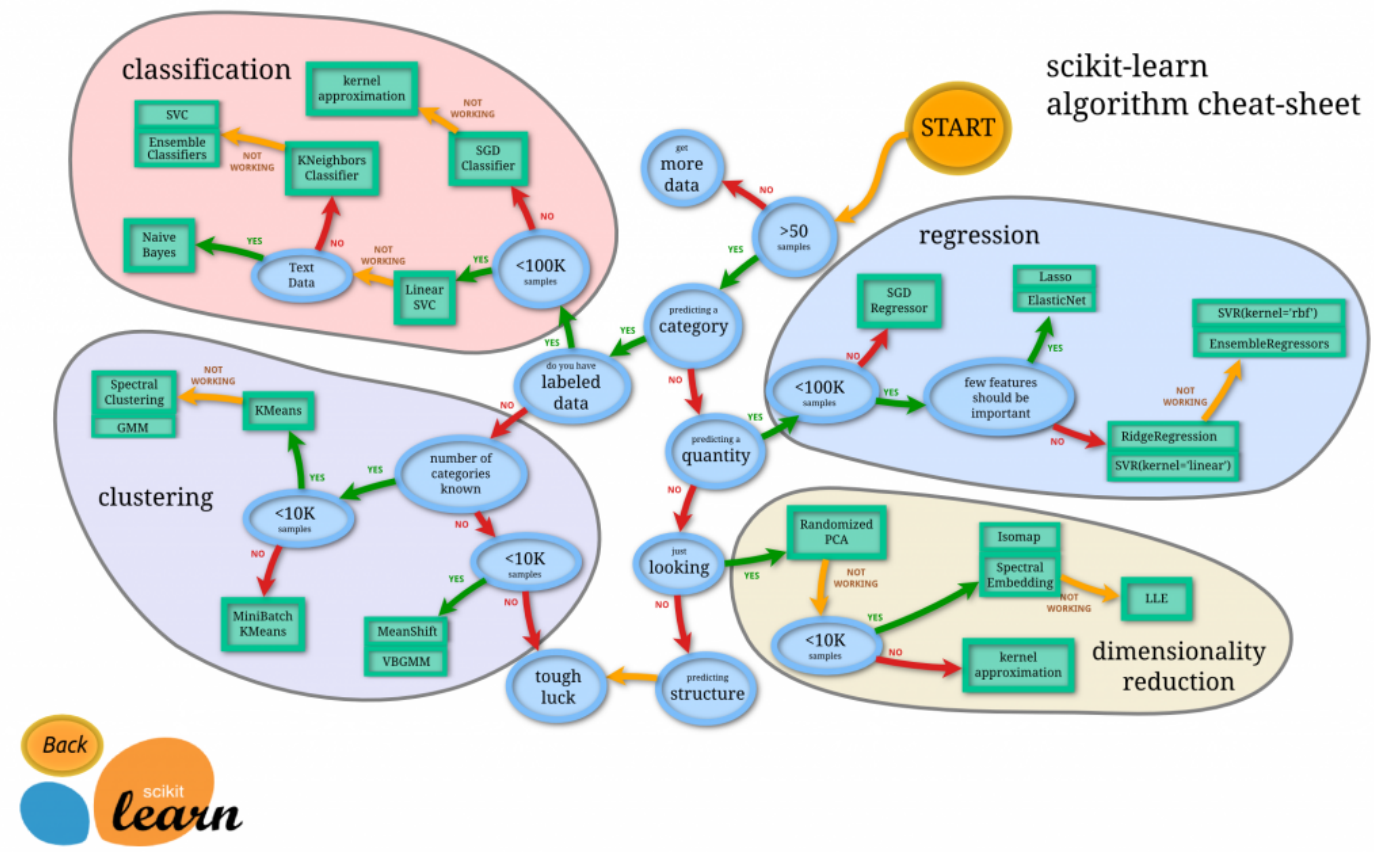


Figure 2 : choix des hyperparamètres pour la SVM

Quels sont les grands classiques ?

Pour beaucoup de problèmes de machine learning, il existe un ou plusieurs algorithmes classiques à essayer en priorité. Pour aider à faire ce choix, le projet Scikit-Learn a édité un arbre de décision très pratique :



À propos de l'auteur: Michel Blancard

Tags: Datasciences Energy Machine-learning OpenSolarMap